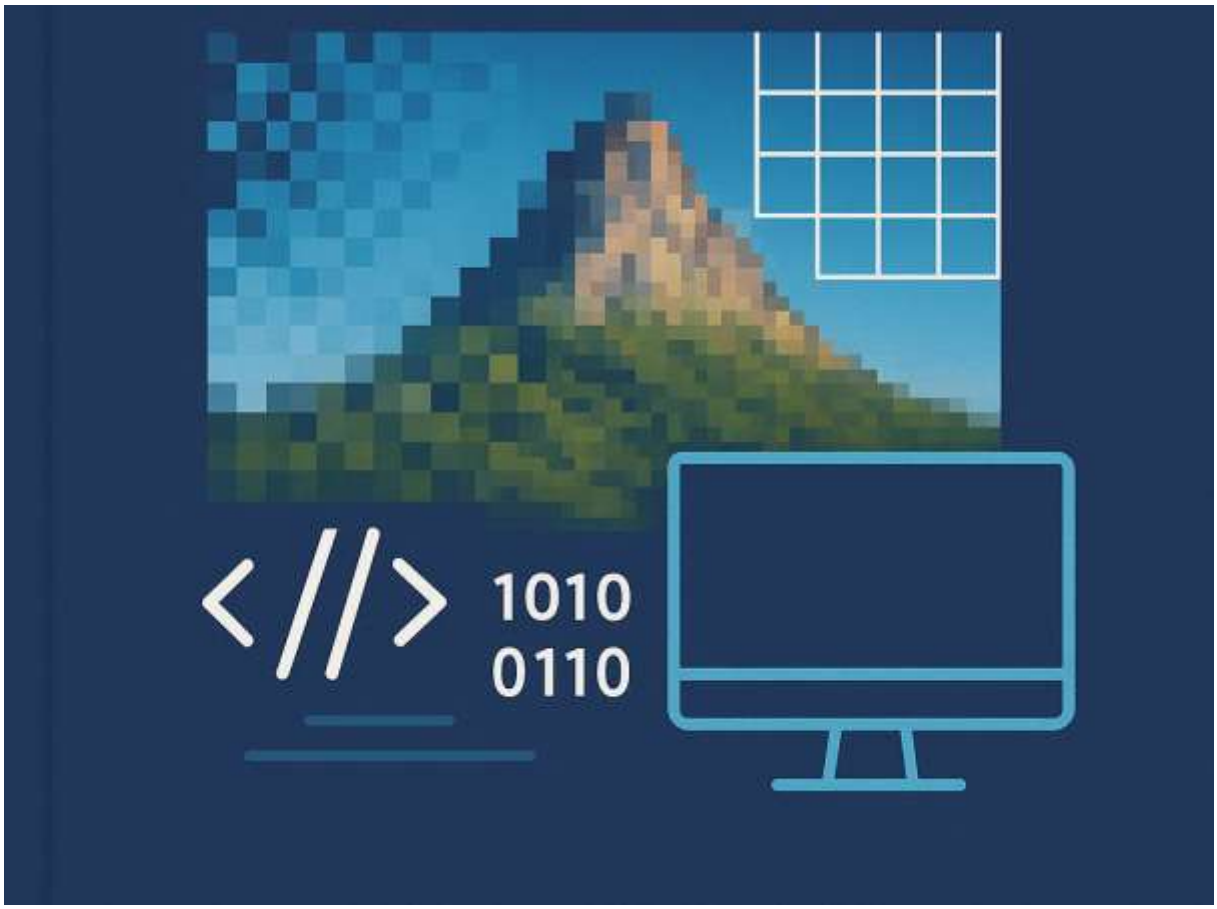
В. П. Майданюк, О. Н. Романюк

Теорія та програмне забезпечення кодування зображень



Міністерство освіти і науки України
Вінницький національний технічний університет

В. П. Майданюк, О. Н. Романюк

Теорія та програмне забезпечення кодування зображень

Електронний навчальний посібник

Вінниця
ВНТУ
2026

УДК 004.921
М90

Рекомендовано до видання Вченою радою Вінницького національного технічного університету Міністерства освіти і науки України (протокол № 3 від 30 вересня 2025 р.)

Рецензенти:
О. Д. Азаров, доктор технічних наук, професор
А. М. Мельник, доктор технічних наук, професор
О. К. Колесницький, кандидат технічних наук, професор

Майданюк, В. П.

М90 Теорія та програмне забезпечення кодування зображень: навчальний посібник [Електронний ресурс] / В. П. Майданюк, О. Н. Романюк. – Вінниця : ВНТУ, 2026. – (PDF, 109 с.)
ISBN 978-617-8163-79-2 (PDF)

Посібник присвячений матеріалам лекційного курсу з дисципліни «Теорія та програмне забезпечення кодування зображень» для аспірантів, що навчаються за спеціальністю «Інженерія програмного забезпечення» денної та заочної форм навчання.

Мета посібника – надати аспірантам можливість більш детально вивчити аудиторний матеріал, опрацювати теми, відведені на самостійну роботу і підготуватися до заліку, а також застосувати отримані знання для подальшої фахової роботи.

Перелік та зміст тем відповідає програмі вказаної вище дисципліни.

УДК 004.921

ISBN 978-617-8163-79-2 (PDF)

ЗМІСТ

ВСТУП

Технологія обробки цифрових зображень досягла в останні роки великих успіхів завдяки тому, що цифрові методи обробки зображень мають ряд переваг перед аналоговими [1, 2, 3]. Однак, значною проблемою є великі розміри файлів, які підлягають передачі та обробці. Наприклад, швидкість передачі символів об'єднаного цифрового потоку під час передавання кольорових телевізійних зображень цифровими методами в форматі 4:2:2 [4] становитиме:

$$Q_Я = 13{,}5\ Мгц * 8\ біт = 108\ Мбіт/с$$

$$Q_К = 6{,}75\ МГц * 8\ біт = 54\ Мбіт/с,$$

$$Q_Σ = Q_Я + 2Q_К = 108\ Мбіт/с + 108\ Мбіт/с = 216\ Мбіт/с,$$

де $Q_Я$ – швидкість передачі символів цифрового сигналу яскравості,

$Q_К$ – швидкість передачі символів різницевих кольорових сигналів.

Передати такий цифровий потік по лініях зв'язку, що існують, вкрай важко. Для зберігання тільки одного телевізійного кадра в цифровій формі необхідно близько 0,5 Мбайта пам'яті, а для однохвилинного репортажу 750 Мбайтів.

Вирішенням проблеми є зменшення обсягу файлів шляхом стиснення даних, або іншими словами кодування зображень.

Актуальність стиснення зображень особливо зросла за останні десять років в зв’язку з появою розвинутих комп’ютерних засобів відображення зображень, поданих в цифровій формі. Це призвело до широкого застосування зображень в комп’ютерних системах та мережах.

Виділення кодування зображень в окремий розділ пов’язано з тим, що зображення – це особливий вид даних, орієнтований на зорове сприйняття людиною, що дозволяє створити спеціальні алгоритми стиснення, які призначені тільки для зображень. Крім того, зображення має надлишковість в двох вимірах, тобто це двовимірний сигнал, що також дає додаткові можливості для стиснення. Завдяки цьому алгоритми стиснення зображень можуть мати дуже високі характеристики.

1 ОСНОВИ КОДУВАННЯ ЗОБРАЖЕНЬ

1.1 Класи зображень

Растрові зображення являють собою двовимірний масив чисел. Елементи цього масиву називають пікселами (від англійського pixel – picture element). Всі зображення можна поділити на дві групи – з палітрою та без неї. В зображень з палітрою в пікселі зберігається число – індекс в деякому одновимірному векторі кольорів, що називається палітрою. Найчастіше зустрічаються палітри з 16 та 256 кольорів.

Зображення без палітри можуть бути в будь-якій системі подання кольорів та в градаціях сірого (grayscale). Для останніх значення кожного піксела інтерпретується як яскравість відповідної точки. Зустрічаються зображення з 2, 16 та 256 рівнями сірого. За використання певної системи подання кольорів кожен піксел являє собою структуру, полями якої є компоненти кольору. Найбільш поширеною є система RGB, в якій колір подано значеннями інтенсивності червоної (R), зеленої (G) та синьої (B) компонент. Існують й інші системи подання кольорів, такі як CMYK, CIE XYZccir60-1 і т. п.

Для того, щоб коректно оцінювати ступінь стиснення, введемо поняття класу зображень. Під класом будемо розуміти деяку сукупність зображень, застосування до якої алгоритму архівації дає якісно однакові результати. Наприклад, для одного класу алгоритм дає дуже високу ступінь стиснення, для другого – майже не стискає, для третього – збільшує розмір файлу.

Розглянемо такі приклади неформального визначення класів зображень [4]:

1. Клас 1. Зображення з невеликою кількістю кольорів (4–16) і великими областями, заповненими одним кольором. Плавні переходи кольорів відсутні. Приклади: ділова графіка – гістограми, діаграми, графіки і под.
2. Клас 2. Зображення з плавними переходами кольорів, побудовані на комп’ютері. Приклади: графіка презентацій, ескізні моделі в САПР, зображення, побудовані за методом Гуро.
3. Клас 3. Фотореалістичні зображення. Приклад: відсканованні фотографії.
4. Клас 4. Фотореалістичні зображення з накладанням ділової графіки. Приклад: реклама.

Розвиваючи таку класифікацію, як окремі класи можуть бути запропоновані неякісно відсканованні в 256 градацій сірого кольору сторінки книг або растрові зображення топографічних карт. Формально будучи 8- або 24-бітовими, вони несуть навіть не растрову, а чисто векторну інформацію. Окремі класи можуть формувати і зовсім специфічні зображення: рентгенівські знімки або фотографії в профіль і фас з електронного досьє [4].

Актуальною залишається задача пошуку найкращого алгоритму для конкретного класу зображень.

Дослідження, які виконуються в області стиснення зображень, орієнтовані насамперед на зображення класу 3 – фотореалістичні зображення, оскільки їх стиснення потребує найбільших витрат і під час його виконання виникають найбільші труднощі. Дійсно, стиснення, наприклад, комп'ютерної графіки може бути легко виконано передачею файлу, що використовується для формування графічного зображення, стиснення зображень з малим числом градацій яскравості також не становить значних труднощів, оскільки для його стиснення можуть бути використані алгоритми з малою обчислювальною складністю.

1.2 Джерела надмірності зображень

В зображенні розрізняють два основних види наддмірності [4]:

- статистична надмірність;
- фізіологічна надмірність.

Перша пов'язана з тим, що будь-які величини, отримані із зображення, не є випадковими. Сусідні відліки часто мають подібні значення яскравості, в чому проявляється важлива властивість їх просторової кореляції. Якщо відповідним чином використати цю властивість, то можна значно зменшити число бітів для подання зображення у цифровій формі.

Фізіологічна надлишковість пов'язана з тією частиною інформації, яка не сприймається оком людини. Скорочення фізіологічної надлишковості значною мірою скорочує й статистичну надлишковість і навпаки.

Необхідно відзначити і так звану афінну надлишковість, на якій базуються фрактальні методи [4] стиснення зображень.

Детальний аналіз різних типів надмірності (просторової, часової, кодової, візуальної тощо) показує, що стиснення зображень є не однією проблемою, а складним, багатогранним інженерним завданням [5–7]. Кожен конкретний тип надмірності потребує окремого алгоритмічного підходу для ефективного використання.

1.3 Класи програм і вимоги до алгоритму стиснення зображень

Розглянемо таку класифікацію програм, що використовують алгоритми архівації [4]:

1. Клас 1. Характеризуються високими вимогами до часу архівації та розархівації. Нерідко потрібен перегляд зменшеної копії зображення та пошук в базі даних зображень. Приклади: видавничі системи в широкому розумінні цього слова. Причому, як системи, де необхідні публікації (журнали) з високою якістю зображень і використанням алгоритмів стиснення без втрат, так і видавничі системи для газет і інформаційних вузлів WWW – де є можливість оперувати зображеннями

меншої якості і використовувати алгоритми стиснення з втратами. В подібних системах доводиться мати справу з повнокольоровими зображеннями різного розміру (від 640×480 – формат цифрового фотоапарата, до 3000×2000) і з великими двокольоровими зображеннями. Оскільки ілюстрації займають велику частину від загального обсягу матеріалу в документі, проблема зберігання стоїть дуже гостро. Проблеми також створює велика різнорідність ілюстрацій (доводиться використовувати універсальні алгоритми). Єдине, що можна сказати завчасно, це те, що переважно це будуть фотореалістичні зображення та ділова графіка.

2. Клас 2. Характеризується високими вимогами до ступеня архівації та часу розархівації. Час архівації значення не має. Іноді подібні програми також потребують від алгоритму стиснення можливості масштабування зображення під конкретну роздільну здатність монітора користувача. Приклад: довідники та енциклопедії на CD-ROM. З появою великої кількості комп'ютерів, оснащених цим приводом (у США – у 50% машин) досить швидко сформувався ринок програм, що випускаються на лазерних дисках. Не дивлячись на те, що ємність одного диска досить велика (приблизно 650 Мбайт), її, як правило, не вистачає. Під час створення енциклопедій та ігор велику частину диска займають зображення та відео. Таким чином, для цього класу програм актуальність мають суттєво асиметричні за часом алгоритми (симетричність за часом – відношення часу архівації до часу розархівації.
3. Клас 3. Характеризується дуже високими вимогами до ступеня архівації. Приклад: нова широко розповсюджена система «Всесвітня інформаційна павутина» – WWW. В цій системі достатньо активно використовуються ілюстрації. Під час оформлення інформаційних або рекламних сторінок хочеться зробити їх більш яскравими та якісними, що звичайно впливає на розмір зображень. Найбільше у цьому випадку страждають користувачі, підключені до мережі за допомогою повільних каналів зв'язку. Якщо сторінка WWW перенасичена графікою, то очікування її повної появи на екрані може затягнутися. Оскільки в цьому разі навантаження на процесор мале, то тут можуть знайти застосування ефективні складні алгоритми стиснення з порівняно великим часом розархівації.

Можна навести багато більш вузьких класів програм. Так своє застосування машинна графіка знаходить і в різноманітних інформаційних системах. Наприклад, вже стає звичним досліджувати ультразвукові та рентгенівські знімки не на папері, а на екрані монітора. Поступово до електронного вигляду переводять і історії хвороб. Зрозуміло, що зберігати ці матеріали логічно в єдиній картотеці. При цьому без використання спеціальних алгоритмів велику частину архівів займуть фотографії. Тому під час створення ефективних алгоритмів розв'язання цієї задачі потрібно

врахувати специфіку рентгенівських знімків – наявність великої кількості областей в зображенні з плавними переходами.

В геоінформаційних системах - при зберіганні аерофотознімків - специфічними проблемами є великий розмір зображення і необхідність вибірки лише частини зображення за вимогою. Крім цього, може знадобитися масштабування. Це, безумовно, накладає свої обмеження на алгоритм стиснення.

В електронних картотеках і досьє різних служб для зображень характерна подібність між фотографіями в профіль та подібність між фотографіями в фас, яку також потрібно враховувати під час створення алгоритму архівації. Подібність між фотографіями спостерігається і в інших спеціалізованих довідниках. Як приклад можна навести енциклопедії птахів або квітів.

Фактично саме характер використання зображень задає тон в виборі того чи іншого алгоритму стиснення. І при цьому задається ступінь важливості таких суперечливих вимог до алгоритму:

1. Високий коефіцієнт стиснення (архівації). Актуальний далеко не для всіх програм. Деякі алгоритми дають кращу якість зображення у разі високих коефіцієнтів стиснення, однак програють іншим алгоритмам у разі низьких.
2. Висока якість зображень. Виконання цієї вимоги напряму суперечить виконанню попередньої.
3. Висока швидкість архівації. Ця вимога для деяких алгоритмів з втратами інформації є взаємновиключною з першими двома. Інтуїтивно зрозуміло, що чим більше часу ми будемо аналізувати зображення, намагаючись отримати найбільший ступінь архівації, тим кращим буде результат. І, відповідно, чим менше часу витрачається на архівацію (аналіз), тим нижчою буде якість зображення і більшим його розмір.
4. Висока швидкість розархівації. Досить універсальна вимога, актуальна для багатьох програм. Однак можна навести приклади програм, де час виконання розархівації далеко на критичний.
5. Масштабування зображень. Ця вимога означає легкість зміни розмірів зображення до розмірів вікна активної програми. Справа в тому, що одні алгоритми дозволяють легко масштабувати зображення прямо під час розархівації, в той час як інші не тільки не дозволяють легко масштабувати, але і збільшують імовірність появи неприємних спотворень після застосування стандартних алгоритмів масштабування до розархівованого зображення. Наприклад, можна навести приклад «поганого» зображення для алгоритму JPEG – це зображення з досить дрібним регулярним рисунком (дрібна клітинка). Характер внесених алгоритмом JPEG змін такий, що зменшення або збільшення зображення може призвести до появи неприємних ефектів.

6. Можливість показати зображення низької розподільчої здатності, використавши тільки початок файлу. Ця можливість актуальна для різного роду мережевих програм, де зчитування зображень може зайняти досить багато часу, і бажано, отримавши початок файлу, коректно показати preview.
7. Стійкість до помилок. Ця вимога означає локальність порушень в зображенні у випадку зруйнування або втрати фрагмента файлу, що передається. Така можливість використовується під час передачі за багатьма адресами зображень в мережі, тобто в тих випадках, коли неможливо використати протокол передачі, який повторно дає запит на дані у сервера у разі виникнення помилок. Наприклад, якщо передається відеоряд кадрів, то було б неправильно використовувати алгоритм, в якого збій приводив би до зупинки правильного показу всіх наступних кадрів. Ця вимога суперечить високому ступеню архівації, оскільки інтуїтивно зрозуміло, що ми маємо вводити в потік надлишкову інформацію. Однак для різних алгоритмів обсяг цієї надлишкової інформації може суттєво відрізнятися.
8. Врахування специфіки зображення. Більш високий ступінь архівації для класу зображень, які статистично частіше будуть застосовуватись в нашій програмі.
9. Редагованість. Під редагованістю розуміється мінімальний ступінь погіршення якості зображення за його повторного зберігання після редагування. Багато алгоритмів з втратою інформації можуть суттєво спотворити зображення за декілька ітерацій редагування.
10. Ефективність програмно-апаратної реалізації.

Ці вимоги до алгоритму реально висувають не тільки виробники ігрових приставок, але і виробники багатьох інформаційних систем [6].

1.4 Метрики оцінення якості зображення

Оцінення якості стиснених зображень є критично важливим для розуміння ефективності алгоритмів та їх впливу на сприйняття користувачем.

Задача відновлення зображення після кодування полягає в тому, щоб одержати вихідне зображення, котре якнайменше відрізняється від вхідного зображення.

Поширеною мірою правильності кодування зображень є усереднена середньоквадратична помилка [1-4]:

$$e^2 = \frac{1}{M*N} \sum_{i=1}^{N} \sum_{J=1}^{M} E(U_{ij} - \hat{U}_{ij})^2, \qquad (1.1)$$

де U_{ij} – значення відліків початкового зображення,

$\hat{U}_{ij}$ – значення відліків відновленого зображення;

M, N – розміри сторін зображення;

E(•) – математичне сподівання.

В експериментах мірою середньоквадратичної помилки служить середнє значення поелементних середньоквадратичних помилок:

$$\bar{e}^2 = \frac{1}{M*N} \sum_{i=1}^{N} \sum_{J=1}^{M} (U_{ij} - \hat{U}_{ij})^2 . \qquad (1.2)$$

На основі наведених вище залежностей можна визначити відношення сигнал/шум:

$$SNR = 10 \lg \frac{255^2}{e^2} \qquad (1.3)$$

У чисельнику задано розмах значень відеоданих, що звичайно задаються у вигляді дискретних відліків, проквантованих на 256 рівнів.

Критерій середньоквадратичної помилки – природна міра спотворень з фізичного і математичного поглядів. Але якщо зображення призначені для візуального спостереження, то перевага цьому критерію віддається не завжди. Це пов'язано з тим, що зорова система не обробляє зображення елемент за елементом, а витягає з нього в процесі нейронного кодування деякі просторові, часові ознаки, а також ознаки кольору.

Найбільш поширеним і найнадійнішим способом визначення якості зображення є суб'єктивна експертиза [4].

Як експертів рекомендується залучати спостерігачів-неспеціалістів, їх оцінки визначають якість зображення саме так, як його сприймає "середній спостерігач".

Відповідно до рекомендації 654 МККР рекомендується п’ятибальна шкала абсолютних оцінок:

Якість	**Погіршення**
5. Відмінна	5. Непомітне
4. Добра	4. Помітне, але не заважає
3. Задовільна	3. Злегка заважає
2. Погана	2. Заважає
1. Дуже погана	1. Дуже заважає

За результатами експертних оцінок звичайно визначається середній бал:

$$\bar{C} = \sum(n_k C_k) / \sum n_k , \qquad (1.4)$$

де n_k – число зображень, віднесених до *k*-ої категорії, а C_k – відповідний їй бал.

В загальному випадку всі метрики поділяють об'єктивні та перцептивні метрики [8].

Об'єктивні метрики

- **Пікове відношення сигнал/шум (PSNR):** Широко використовувана об'єктивна метрика, яка кількісно визначає середню попіксельну

різницю між вихідним і стисненим кадром. Вона є швидкою та простою в обчисленні. Вище значення PSNR зазвичай означає кращу якість, але воно не завжди відповідає людському сприйняттю.

- **Індекс структурної подібності (SSIM):** Розроблений для вимірювання сприйнятої якості шляхом порівняння трьох фундаментальних аспектів зображення: яскравості, контрасту та структури. На відміну від PSNR, SSIM моделює, як людський зір сприймає якість зображення, що робить його кращим провісником суб'єктивної якості, ніж PSNR. Він також чутливіший до розмиття, артефактів стиснення та втрати деталей.

Таблиця 1.1 – Огляд метрик якості зображення та їх застосування

Метрика	**Тип**	**Що вимірює**	**Переваги**	**Недоліки**	**Найкращі сценарії використання**
PSNR	Об'єктивна	Середні попікселні різниці	Швидкий та легкий в обчисленні	Не завжди відповідає людському сприйняттю	Швидкі перевірки, виявлення екстремальних спотворень
SSIM	Об'єктивна	Структурна подібність (яскравість, контраст, структура)	Ближчий до людського сприйняття, чутливий до структурних спотворень	Більш обчислювально інтенсивний, ніж PSNR	Коли важлива структурна цілісність, оцінка артефактів стиснення
VMAF	Перцептивна	Як люди сприймають якість відео/зображення (на основі ML)	Високоточна, основана на сприйнятті оцінка, враховує машинне навчання	Потребує більше обчислю-вальних ресурсів, специфічна для відео	Порівняння налаштувань кодування, оптимізація адаптивного потокового передавання

Перцептивні метрики

- **Video Multimethod Assessment Fusion (VMAF):** Розроблений Netflix, VMAF спеціально призначений для прогнозування того, як люди насправді сприймають якість відео. Він містить машинне навчання (ML) для розширеної, заснованої на сприйнятті оцінки. Застосування VMAF є ідеальним для порівняння налаштувань кодування, оптимізації адаптивного потокового передавання бітрейту та забезпечення найкращого досвіду перегляду.
- **Комбіноване використання метрик:** Багато робочих процесів обробки відео інтегрують усі три метрики (PSNR, SSIM, VMAF) для забезпечення збалансованої оцінки як математичної точності, так і людського візуального досвіду.

1.5 Етапи кодування зображень

Ефективне кодування зображень звичайно виконується за три етапи:

1. На першому етапі знаходиться подання зображення у вигляді набору коефіцієнтів деякого перетворення. Ця операція, як правило, обернена.
2. На другому етапі зменшується точність подання компонент зображення, але так, щоб виконувались задані вимоги до якості зображення. Така операція призводить до втрат інформації, тому не є оберненою.
3. На третьому етапі усувається статистична надлишковість в зображенні, отриманому після виконання перших двох етапів. Для виконання цього етапу може застосовуватись кодування Гаффмана, арифметичне кодування та інші. Ця операція обернена [4].

Найбільш інтенсивні дослідження виконуються із пошуку нових методів для виконання першого і другого етапів, оскільки у цьому випадку витрачається найбільше обчислювальних ресурсів і дослідження пов’язані не тільки з пошуком математичного перетворення, але і з дослідженням особливостей зорового сприйняття зображення і особливостей завадостійкої передачі цього зображення по каналах зв’язку.

1.6 Аналіз основних методів кодування зображень

Базовим методом цифрового кодування джерел зображень є імпульсно-кодова модуляція (ІКМ). Вона характеризується тим, що кожному закодованому в цифрову форму слову відповідає квантований в часі і за амплітудою відлік відеоінформації. Водночас мають виконуватись вимоги теореми дискретизації $f_д > 2W_0$, де W_0 максимальна частота, яка міститься в сигналі. Щоб запобігти появі фальшивих контурів на однокольоровому зображенні необхідно більше 50 рівнів квантування, що відповідає 6- ÷ 8-розрядному кодовому слову на кожний елемент (піксел) зображення. Через великі обсяги інформації ІКМ застосовується лише у разі внутрістудійної передачі телевізійних зображень паралельним кодом і як базове канонічне подання зображення у цифровій формі [1,2,3,4].

Серед методів кодування з передбаченням найбільш досліджена диференційно-імпульсна кодова модуляція (ДІКМ). Суть ДІКМ така. Організується передбачення значення яскравості кожного наступного елемента зображення на основі лінійної комбінації значень яскравості попередніх елементів. Оцінка, отримана внаслідок передбачення, віднімається від істинного значення яскравості елемента зображення і різницевий сигнал квантується невеликим числом рівнів. За рахунок цього і досягається скорочення обсягу даних [4].

Методи кодування зображень з передбаченнями дозволяють стиснути зображення в 2 – 2,5 рази за простої технічної реалізації. Серед недоліків цих методів необхідно відзначити такі:

- похибки в місцях різких перепадів яскравості;
- низька завадостійкість.

Інтерполяційні методи основані на числових методах апроксимації, за допомогою яких послідовність або двовимірний масив відліків яскравості наближено подаються через неперервні функції [4]. У цьому разі кодуються лише окремі відліки зображення, а сусідні з ними отримують як результат інтерполяції поліномами, звичайно, не більше ніж третього степеня. Коефіцієнт стиснення, який досягається за умови використання цих методів, дорівнює 5 – 6.

Основним недоліком інтерполяційних методів є великий обсяг обчислень під час інтерполяції поліномами високих степенів, а також необхідність зберігання координат базових відліків зображення.

Важливий клас методів кодування зображень – це методи кодування на основі перетворень. Кодування на основі перетворень – непрямий метод. Зображення піддаються унітарному математичному перетворенню, коефіцієнти, отримані внаслідок перетворення, квантуються і кодуються для передачі по каналу зв'язку або запису в файл. Для більшості зображень значення багатьох коефіцієнтів перетворення порівняно мале. Такі коефіцієнти часто можна відкинути або виділити для їх кодування невелике число двійкових розрядів. Найчастіше використовують двовимірні ортогональні перетворення. Це такі як перетворення Карунена–Лоева, дискретне перетворення Фур'є (ДПФ), дискретне косинусне перетворення (ДКП), перетворення Уолша–Адамара [2, 3, 8, 9, 10] та інші. Коефіцієнти стиснення, за застосування цих методів, можуть досягати 6–8, завдяки чому деякі з цих методів (ДКП) знайшли широке застосування як основа промислових стандартів з кодування зображень. Спільним недоліком цих методів є досить висока обчислювальна складність, а також неможливість розв'язання ряду задач обробки зображень на скороченому обсязі даних.

Серед статистичних методів найбільш широке застосування знаходять блочні методи кодування зображень. Блоки розміром M×N елементів кодуються згідно з імовірністю їх появи. Для найбільш імовірних конфігурацій використовуються короткі кодові слова, а для менш імовірних – довгі кодові слова (алгоритм Гаффмана), внаслідок чого досягається стиснення даних [4]. Коефіцієнти стиснення у випадку використання цих методів можуть досягати 4–5.

Перспективним для кодування як рухомих так і нерухомих зображень є метод покомпонентного кодування [4]. Особливістю цього методу є формування декількох двовимірних сигналів, що несуть інформацію про деталі зображення різного розміру. Наявність декількох каналів окремої обробки деталей зображення різного розміру дозволяє ефективно кодувати зображення як внутрікадровими, так і міжкадровими методами з урахуванням особливостей сприйняття інформації зоровим аналізатором людини. Важливою перевагою цього методу є те, що формати подання

даних після стиснення є компонентами вихідного зображення, тобто кожна компонента візуально подає зображення з тим чи іншим ступенем роздільної здатності, а це дозволяє розв'язувати ряд задач обробки зображень на скороченому обсязі даних.

Хоча практично досягнутий коефіцієнт стиснення за застосування цього методу незначно менший порівняно з методами кодування з перетвореннями, його технічна реалізація значно простіша і відповідно швидкодія значно більша.

Для стиснення зображень у реальному часі за сукупністю таких параметрів як простота технічної реалізації, обчислювальна складність, коефіцієнт стиснення найкращі результати одержують у випадку використання Wavelet-кодування, що базується на пірамідальних схемах розкладення початкового зображення на компоненти (S-перетворення) [4, 11]. 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. У випадку великих коефіцієнтів стиснення на різких границях, особливо діагональних, можливі спотворення. Важливою перевагою S-перетворення є можливість показати «огрублене» зображення (низької роздільної здатності), використавши тільки початок файлу.

Найбільші коефіцієнти стиснення забезпечує метод фрактального стиснення зображень [5,15 – 19 та ін. (більш детальну інформацію можна отримати в Internet наприклад за адресою *http://www.dip.ee.uct.ac.za/imageproc/compression/fractal/*)], відкритий в 1988 році. Процес фрактального стиснення оснований на твердженні, що зображення реального світу мають афінну надлишковість. Коефіцієнти стиснення можуть досягати 50-60 раз. Основним недоліком цього методу є велика обчислювальна складність. Однак, враховуючи великий ступінь стиснення, який можна отримати цим методом, а також гігантський прогрес у збільшенні продуктивності мікропроцесорів та інших апаратних засобів, потрібно очікувати найширшого застосування цього методу в найближчі роки.

Останні роки характеризуються зростанням інтересу до застосування штучних нейронних мереж для ущільнення зображень. Ці методи використовують різні типи нейронних мереж, включаючи глибокі нейронні мережі, штучні нейронні мережі, рекурентні нейронні мережі та згорткові нейронні мережі [14,15,16,17]. Глибоке стиснення зображень демонструє кращі результати, ніж традиційні кодеки, такі як JPEG, для природних зображень [15].

1.7 Стандарти стиснення зображень та відео

Міжнародні стандарти стиснення є кульмінацією теоретичних досліджень та інженерних рішень, що забезпечують сумісність та ефективність у різних пристроях та платформах.

JPEG (Joint Photographic Experts Group)

Принципи. JPEG є найпоширенішим стандартом стиснення зображень із втратами [18]. Він розроблений для фотографій та зображень з плавними переходами кольорів.

Конвеєр JPEG

1. Перетворення колірного простору: Зображення перетворюється з RGB на YCbCr, де Y – компонент яскравості (люмінанс), а Cb та Cr – компоненти кольоровості (хроматичність). Людська візуальна система більш чутлива до змін яскравості, ніж до кольору.
2. Хроматична субдискретизація: Компоненти кольоровості (Cb, Cr) субдискретизуються (наприклад, 4:2:0), оскільки людське око менш чутливе до деталей кольору. Це зменшує обсяг даних без значного погіршення сприйнятої якості.
3. Блокове ДКП: Кожен компонент (Y, Cb, Cr) поділяється на блоки 8×8 пікселів, до яких застосовується ДКП.
4. Квантування: Коефіцієнти ДКП квантуються за допомогою таблиць квантування. Це основний етап із втратами, де високочастотні коефіцієнти (менш важливі для сприйняття) відкидаються або сильно округлюються.
5. Ентропійне кодування: Квантовані коефіцієнти кодуються без втрат за допомогою кодування Гаффмана або арифметичного кодування.

Переваги

- Універсальна сумісність: Підтримується майже всіма пристроями та програмним забезпеченням.
- Хороший коефіцієнт стиснення: Забезпечує значне зменшення розміру файлу для фотографій.
- Регульований рівень стиснення: Дозволяє користувачам балансувати якість зображення та розмір файлу.

Недоліки

- Блокові артефакти: Схильність до появи помітних блокових артефактів за високих рівнів стиснення.
- Не підтримує прозорість: Не підтримує альфа-канал для прозорості.

HEVC (High Efficiency Video Coding, H.265)

Принципи. HEVC є стандартом стиснення відео, який значно покращує ефективність стиснення порівняно з попередніми стандартами, такими як H.264/AVC [19]. Він також є основою для формату зображень HEIF (High Efficiency Image Format).

Ключові особливості

- Блоки кодування дерева (CTU): Замінює макроблоки 16x16 пікселів на більші CTU (до 64×64 пікселів), що дозволяє краще адаптуватися до різних областей зображення.
- Розширені перетворення: Використовує як ДКП, так і дискретне синусне перетворення (ДСП) з різними розмірами блоків (від 4×4 до 32×32).
- Покращене прогностичне кодування: Більш складні режими внутрішньокадрового та міжкадрового прогнозування та компенсації руху.
- Підтримка 4K/Ultra HD: Розроблений для ефективного стиснення відео високої роздільної здатності та 16-бітної глибини кольору.

Переваги

- Значно вища ефективність стиснення: Забезпечує від 25% до 50% краще стиснення даних за того самого рівня якості відео порівняно з AVC.
- Вища якість за того самого бітрейту. Дозволяє отримати кращу якість зображення за того самого бітрейту.
- Підтримка HDR та 16-бітного кольору: Важливо для сучасного контенту.

Недоліки

- Вища обчислювальна складність: потребує більше обчислювальної потужності для кодування та декодування.
- Обмежена сумісність: Хоча підтримка зростає, вона все ще не така універсальна, як JPEG.

WebP та AVIF

WebP

- *Тип*. Підтримує як стиснення із втратами, так і без втрат [20].
- *Застосування*. Розроблений Google для веб-зображень, щоб замінити JPEG, PNG та GIF.
- *Переваги*. Кращий коефіцієнт стиснення порівняно з JPEG/PNG, підтримка прозорості та анімації.
- *Недоліки*. Помірна швидкість декодування, не 100% сумісність з усіма браузерами.

AVIF (AV1 Image File Format):

- *Тип*. Підтримує як стиснення із втратами, так і без втрат [20].
- *Застосування*. Заснований на відеокодеку AV1, призначений для веб-зображень та потокового відео.
- *Переваги*. Найкращий коефіцієнт стиснення, підтримка HDR, прозорості, анімації. Використовує більші блоки (до 128x128) та множинні режими прогнозування.
- *Недоліки*. Повільна швидкість декодування, зростаюча, але не повна підтримка браузерами.

JPEG 2000 та Дискретне вейвлет-перетворення (ДВП)

Теоретичні основи ДВП

Дискретне вейвлет-перетворення (ДВП) є ключовою технологією стиснення, що використовується в стандарті JPEG 2000 [21,22]. На відміну від ДКП, яке працює з фіксованими блоками, ДВП забезпечує багатороздільний аналіз, що дозволяє уникнути блокових артефактів та краще обробляти краї зображень. ДВП розкладає сигнал на апроксимаційні (низькочастотні) та детальні (високочастотні) коефіцієнти шляхом проходження через серію фільтрів (низькочастотний та високочастотний фільтри). Після фільтрації сигнали субдискретизуються (зменшуються вдвічі), що зменшує кількість даних. Цей процес може бути застосований рекурсивно до апроксимаційних коефіцієнтів для отримання багаторівневого розкладу.

Етапи стиснення JPEG 2000

1. Підготовка вихідного зображення: Включає перетворення колірного простору та, можливо, DC-зсув.
2. Дискретне вейвлет-перетворення (ДВП): Застосовується до зображення (або його тайлів), розкладаючи його на піддіапазони (LL, LH, HL, HH). Для втратного стиснення використовується фільтр Daubechies 9-tap/7-tap, а для безвтратного — 5-tap/3-tap.
3. Квантування: Коефіцієнти вейвлетів квантуються (для втратного стиснення) на основі їхнього піддіапазону та таблиці квантування.
4. Розклад на бітові площини: Квантовані коефіцієнти розкладаються на бітові площини, починаючи з найбільш значущого біта.
5. Арифметичне кодування: Бітові площини кодуються за допомогою арифметичного кодування для усунення надмірності.
6. Постобробка (для втратного стиснення): Визначає, які бітові потоки об'єднуються в кінцеве зображення.

Переваги JPEG 2000

- Вищий коефіцієнт стиснення: Значно вищі коефіцієнти стиснення порівняно з оригінальним JPEG без компромісів у якості.
- Масштабованість: Дозволяє прогресивне завантаження за якістю, роздільною здатністю або компонентами.
- Безвтратне та втратне стиснення: Підтримує обидва режими в одному бітовому потоці.
- Відсутність блокових артефактів: Завдяки вейвлет-перетворенню.
- Регіони інтересу (ROI): Можливість кодувати певні області зображення з вищою якістю.

Недоліки JPEG 2000

- Вища обчислювальна складність: Вимагає більше обчислювальної потужності для стиснення та декомпресії, що може призвести до затримок.
- Обмежена підтримка: Незважаючи на технічні переваги, не досяг такого значного поширення, як оригінальний JPEG.

Таблиця 1.2 – Порівняння ключових стандартів стиснення зображень

Стандарт/ Формат	Тип стиснення	Основне застосування	Основні алгоритми	Ключові переваги	Ключові недоліки
JPEG	Із втратами	Фотографії, веб-зображення	ДКП, ентропійне кодування (Гаффмана)	Універсальна сумісність, хороший коефіцієнт стиснення	Блокові артефакти, не підтримує прозорість
HEVC (H.265)	Із втратами	Відео, зображення (HEIF)	ДКП, ДСП, CTU (до 64x64), прогностичне кодування	Значно кращий коефіцієнт стиснення, підтримка 4K/Ultra HD, 16-бітний колір	Вища обчислювальна складність, обмежена сумісність
WebP	Із втратами / Без втрат	Веб-зображення	Прогностичне кодування, кодування перетворенням, ентропійне кодування	Кращий коефіцієнт стиснення порівняно з JPEG/PNG, підтримка прозорості та анімації	Помірна швидкість декодування, не повна сумісність з усіма браузерами
AVIF	Із втратами / Без втрат	Веб-зображення, потокове відео	Більші блоки (до 128×128), множинні режими прогнозування, розширене ентропійне кодування	Найкращий коефіцієнт стиснення, підтримка HDR, прозорості, анімації	Повільна швидкість декодування, зростаюча, але не повна підтримка браузерами
JPEG 2000	Із втратами / Без втрат	Медична візуалізація, архівування, CCTV	ДВП, квантування, розклад на бітові площини, арифметичне кодування	Вищий коефіцієнт стиснення, масштабованість, ROI, без блокових артефактів	Вища обчислювальна складність, обмежена сумісність

1.8 Перспективи розвитку кодування зображень

Перспективи розвитку кодування зображень спрямовані на:
- підвищення якості за зменшення бітрейту,
- глибоку інтеграцію зі штучним інтелектом,
- підтримку нових форматів (HDR, 3D, AR/VR),
- безпечність та захист автентичності.

З врахуванням цього можна зробити такі висновки:

1. У найближчі роки провідну роль відіграватимуть нейромережеві методи та універсальні формати (JPEG XL, AVIF), що поєднують ефективність, багатофункційність і сумісність. На відміну від

традиційних алгоритмів стиснення зображень (наприклад, JPEG, PNG), які використовують фіксовані правила для зменшення обсягу даних, нові підходи на основі нейронних мереж і глибокого навчання можуть «вивчати» особливості зображень і знаходити оптимальні способи їх кодування.

2. Великі зусилля будуть спрямовані на дослідження фрактального методу та його модифікацій, що можливо сприятиме появі нового стандарту, основою якого буде фрактальний метод кодування зображень.
3. Кодування зображень також буде розвиватися для задоволення потреб конкретних галузей: медична візуалізація, космічні технології, доповнена та віртуальна реальність (AR/VR), мобільний зв'язок. Тому будуть проводитись пошуки та дослідження нових методів кодування, які можуть забезпечити високий коефіцієнт стиснення і високу якість відновленого зображення, незважаючи на їх обчислювальну складність. Також будуть продовжені пошуки простих методів кодування, будуть досліджуватись комбінації цих методів з урахуванням нових технічних можливостей.

1.9 Запитання для самоконтролю

1. Чому необхідне стиснення зображень?
2. Охарактеризуйте класи зображень.
3. Які основні джерела надлишковості зображень?
4. Охарактеризуйте основні класи програм, які використовують зображення.
5. Наведіть вимоги до алгоритмів кодування зображень та охарактеризуйте їх
6. Які етапи кодування зображень?
7. Охарактеризуйте критерії правильності кодування.
8. Чому основним критерієм якості відновлених зображень є метод експертних оцінок?
9. Наведіть недоліки і переваги інтерполяційних методів кодування зображень.
10. Які ортогональні перетворення застосовуються в процесі кодування зображень?
11. Який вид надлишковості усувається за фрактального стиснення?
12. Охарактеризуйте статистичні методи стиснення зображень.
13. Які види зображень найкраще стискаються за допомогою Wavelet-кодування?
14. Який недолік ДІКМ?

2 АЛГОРИТМИ СТИСНЕННЯ ЗОБРАЖЕНЬ БЕЗ ВТРАТ

2.1 Класифікація алгоритмів стиснення

Серед алгоритмів стиснення без втрат дві схеми стиснення - кодування Гаффмана (Huffman) і LZW - кодування (за початковими літерами прізвищ Лемпел (Lempel) і Зів (Ziv) - його автори і Уелч (Welch), який його суттєво модифікував), формують основу для багатьох систем стиснення. Ці схеми представляють два різних підходи до стиснення даних [23,24]:

- статистичні методи стиснення;
- словарні (евристичні) методи стиснення.

Статистичні алгоритми (Гаффмана, Шеннона-Фано, арифметичні) потребують знання імовірностей появи символів в зображенні, оцінкою якої є частота появи символів у вхідних даних. Як правило, ці імовірності невідомі. З урахуванням цього статистичні алгоритми можна поділити на три класи:

1. Неадаптивні – використовують фіксовані, завчасно задані імовірності символів. Таблиця імовірностей символів не передається разом з файлом, оскільки вона відома завчасно. Недолік: невеликий перелік файлів, для яких досягається прийнятний коефіцієнт стиснення.
2. Напівадаптивні – для кожного файлу будується таблиця частот символів і за її допомогою стискають файл. Разом зі стисненим файлом передається таблиця символів. Такі алгоритми досить непогано стискають більшість файлів, але необхідна додаткова передача таблиці частот символів, а також два проходи початкового файлу для виконання кодування.
3. Адаптивні - починають працювати з фіксованою початковою таблицею частот символів (зазвичай всі символи спочатку рівноімовірні) і в процесі роботи ця таблиця змінюється залежно від символів, що зустрічаються в файлі. Переваги: однопрохідність алгоритму, не потребує передачі таблиці частот символів, досить ефективно стискає широкий клас файлів.

Евристичні (словарні) алгоритми стиснення (типу LZ77, LZ78), як правило, шукають в файлі рядки, що повторюються, і будують словник фраз, що вже зустрічались.

Зазвичай такі алгоритми мають цілий ряд специфічних параметрів (розмір буфера, максимальна довжина фрази і т. п.), підбір яких залежить від досвіду автора роботи, і ці параметри добираються таким чином, щоб досягти оптимального співвідношення часу роботи алгоритму, коефіцієнта стиснення та переліку файлів, що добре стискаються.

2.2 Алгоритм RLE

Цей алгоритм надзвичайно простий в реалізації. RLE (Run Length Encoding), або групове кодування, – один з найбільш давніх алгоритмів кодування графіки. Його суть полягає в заміні повторюваних підряд символів на пару: повторюваний символ та лічильник повторень. Проблема є в тому, щоб архіватор під час відновлення міг відрізнити в результувальному потоці таку кодовану серію від інших символів. Рішення очевидне – помістити у всі ланцюжки деякі заголовки (наприклад, використати перший біт як ознаку кодування серії). Метод достатньо ефективний для графічних зображень в форматі «байт на піксел» (наприклад, формат PCX використовує кодування RLE).

Проте недоліки алгоритму очевидні - це, зокрема, мала пристосованість до багатьох типів файлів, що часто зустрічаються, наприклад, до текстових. Тому його можна ефективно використовувати тільки в поєднанні з вторинним кодуванням. Цей підхід знайшов своє відображення в алгоритмі кодування факсів: спочатку зображення розбивається на чорні та білі точки, які перетворюються алгоритмом RLE в потік довжин серій, а потім ці довжини серій кодуються методом Гаффмана зі спеціально підібраним (експериментально) деревом [24].

2.3 Кодування Гаффмана

Статистичне кодування Гаффмана ставить у відповідність вхідним символам, поданим ланцюжками бітів однакової довжини (наприклад, восьмибітовими байтами), ланцюжки бітів змінної довжини. Довжина коду для символу пропорційна двійковому логарифму його частоти, взятому із протилежним знаком. Це кодування є префіксним, що дозволяє легко його декодувати однопрохідним алгоритмом. Префіксний код зручно відображати у вигляді двійкового дерева, у якого символи знаходяться в листях, а дуги відмічені 0 або 1. Тоді код символу можна задавати як шлях від кореня дерева до листка, який містить цей символ.

Під час використання адаптивного кодування Гаффмана необхідне постійне коректування дерева згідно зі зміною статистики вхідного потоку. В процесі реалізації це звичайно потребує значних витрат на перебалансування кодового дерева згідно з новими частотами символів на кожному кроці. Стиснення інформації виконується за два проходи:

1. Під час першого проходу оцінюються частоти появи символів в початковому файлі.
2. Під час другого проходу виконується подання цих символів префіксними кодами змінної довжини.

Як це реалізується на практиці, розглянемо на прикладі деякого файлу. Після першого етапу отримано таку інформацію про частоти появи символів в файлі:

Символ	U	A	M	H	N	F
Частота	43	19	7	23	27	61

Обсяг файлу 180 байт (1440 біт). Він складається з шести різних символів, які зустрічаються в файлі від 7 до 61 раза.

На наступному етапі необхідно закодувати символи згідно з частотою їх повторення. Цей процес пояснює рис. 2.1. Виберемо із таблиці два символи з найменшою частотою появи у вхідному файлі і утворимо з них вузол. Вміст вузла - це сума обох частот символів (19+7=26). Далі знову знаходимо два символи з найменшою частотою. При цьому частоти 19 і 7 вже не враховуються, але в пошуку символів з найменшими частотами бере участь утворений вузол. Разом з символом «H» перший вузол створює наступний вузол (26+23=49).

Якщо проводити вибір символів за такою схемою, то колись структура має закінчитись. Для останнього вузла сума вже не записується, оскільки порівняння закінчилось. Цей останній вузол називається коренем, через який можливий вхід в дерево кодування.

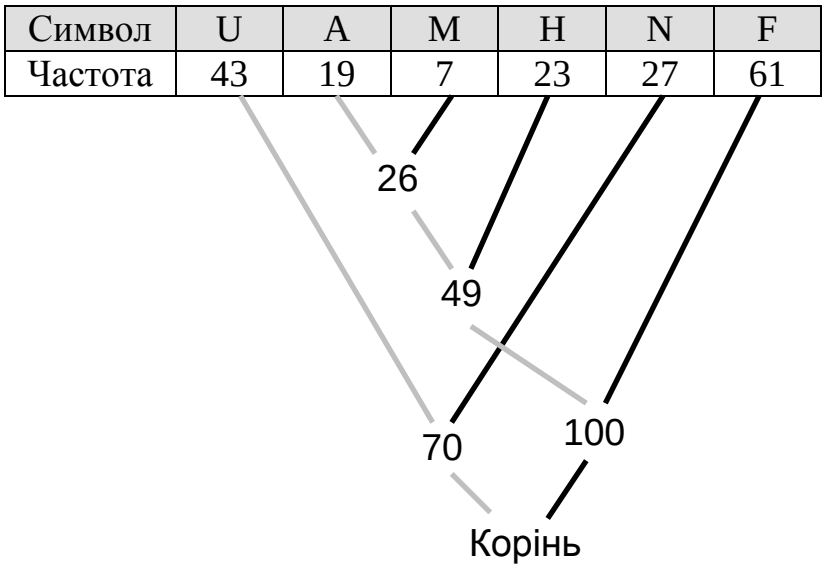


Рисунок 2.1 – Побудова двійкового дерева

Новий код для символів упакованого файлу одержимо із створеного двійкового дерева. Ця структура називається двійковим деревом тому, що кожний вузол утворюється із двох можливих розгалужень. Оскільки для біта існують теж тільки дві можливості (0 і 1), то утворюється новий код, який, починаючи від кореня, проходить від вузла до вузла, до відповідного символу. Для кожного вузла необхідно вирішити, куди потрібно повернути (ліворуч чи праворуч), щоб дійти до символу, який кодується. Будемо вважати, що ліва гілка відповідає «0», а права – «1».

Перехід від кореня дерева до будь-якого символу дає такі коди:

U	Ліворуч-ліворуч	00
A	Праворуч-ліворуч-ліворуч-ліворуч	1000
M	Праворуч-ліворуч-ліворуч-праворуч	1001
H	Праворуч-ліворуч-праворуч	101
N	Ліворуч-праворуч	01
F	Праворуч-праворуч	11

Символи, які найчастіше зустрічаються («U» і «F») отримали короткі 2-бітові кодові комбінації, а ті, що найрідше («M») – 4-бітові кодові комбінації. Ступінь стискання показано в табл. 2.1: початкові 1440 біт файлу стиснулись до 435 біт, що відповідає ступеню стискання 31%.

Таблиця 2.1 – Оцінка ступеня стискання файлу

Символ	**Частота**	**Кількість бітів на символ**	**Загальна кількість**
U	43	2	86
A	19	4	76
M	7	4	28
H	23	3	69
N	27	2	54
F	61	2	122
Всього	**180**		**435**

На останньому етапі програма оброблює файл і замінює кожний символ на його код. Також вона записує кодове дерево у файл, оскільки тільки за допомогою цього дерева можна знову декодувати файл. Ступінь стискання погіршується через те, що і кодове дерево потребує додаткової пам’яті.

Таким чином, алгоритм статистичного кодування способом Гаффмана буде складатись з таких етапів:

1. Визначення кількості повторювання кожного символу.
2. Побудова двійкового дерева.
3. Побудова кодової послідовності для кожного символу.
4. Заміна символів заданого файлу відповідними кодовими послідовностями і запис результату.

Недоліком кодування Гаффмана є залежність ступеня стискання від близькості імовірностей символів до від’ємних степенів двійки, а також складна апаратурна реалізація, яка пов’язана з необхідністю запам’ятовування кадру зображення, тобто кодування виконується за два проходи.

2.4 Кодування за ступенем новизни

За невідомого розподілу частот використовують простий і досить ефективний метод стискання – кодування за ступенем новизни. Цей метод був так названий Елайєсом в 1987 році, але він був відкритий ще раніше Б. Я. Рябком в 1980 році і відомий під назвою стискання за допомогою купки книжок [24].

Ідея методу така: нехай алфавіт джерела складається з N символів з номерами 1, 2, ..., N. Кодер зберігає список символів, які являють собою деяку перестановку алфавіту. Коли на вхід надходить символ C, який має в списку номер «i», кодер передає код номеру «i». Потім кодер переставляє символ C на початок списку, збільшуючи на одиницю номери всіх символів, які стоять перед C. Таким чином, більш «популярні» символи будуть тяжіти до початку списку та будуть мати більш короткі коди.

Як код для кодування за ступенем новизни можна використовувати монотонний код – універсальний код джерела, для якого відомо лише впорядкованість імовірностей символів. Монотонний код побудований так, що більш близькі до початку списку позиції отримують більш короткі коди. Результат – літери, які часто з'являються та тяжіють до початку списку, будуть мати короткі кодові позначення. Розглянемо стискання інформації методом кодування за ступенем новизни на прикладі.

Нехай необхідно передати слово w=221312233 в алфавіті A={1,2,3}. Монотонним кодом позиції 1 є 0, позиції 2 - 10, позиції 3 - 11. Купка книжок за послідовної появи літер слова w змінюється таким чином:

(1,2,3) -- (2,1,3) -- (2,1,3) -- (1,2,3) -- (3,1,2) і т.д.

Кодом слова буде

100101110110…

Кодування за ступенем новизни ненабагато гірше найкращого кодування методом Гаффмана. Цей метод поступається кодуванню Гаффмана у випадку стискання текстових файлів, однак, для файлів з малим початковим алфавітом, можна досягти гарних результатів. Саме такими файлами є файли, наприклад, контурних зображень. Кодування за ступенем новизни не потребує попереднього перегляду початкового масиву. Цей метод легко пристосовується до можливих коливань частот, тому він названий ще *локально адаптивним*.

2.5 Алгоритм LZW

Власне вихідний Lempel/Ziv підхід до стиснення даних був вперше обнародуваний в 1977 р., а вдосконалений (Terry Welch) варіант був опублікований в 1984 р. [4,24]. Алгоритм дуже простий. Якщо коротко, то LZW - стиснення заміняє рядки символів деякими кодами. Це робиться без будь-якого аналізу вхідного тексту. Замість цього, у разі додання кожного нового рядка проглядається таблиця рядків. Стиснення відбувається, коли

код заміняє рядок символів. Коди, генеровані LZW-алгоритмом, можуть бути будь-якої довжини, але вони мають містити більше бітів, ніж одиничний символ. Перші 256 кодів (коли використовуються 8-бітові символи) за замовчуванням відповідають стандартному набору символів. Решта кодів відповідають рядкам, що обробляються алгоритмом.

Під час стиснення та відновлення LZW маніпулює трьома об`єктами: потоком символів, потоком кодів і таблицею рядків. Під час стиснення потік символів є вхідним і потік кодів - вихідним. У випадку відновлення вхідним є потік кодів, а потік символів – вихідним. Таблиця рядків породжується і за стиснення, і за відновлення, однак вона ніколи не передається від стиснення до відновлення і навпаки.

Оскільки ми не знаємо наперед обсяг файлу, то важко визначити, яка розрядність кодів буде оптимальною. Тому доцільно використовувати коди різної довжини.

2.5.1 Стиснення інформації

Наведемо алгоритм в найпростішій формі. Кожний раз, коли генерується новий код, новий рядок додається в таблицю рядків. LZW постійно перевіряє, чи цей рядок вже відомий, і, якщо так, виводить код без генерування нового.

Процедуру LZW-стиснення наведено на рис. 2.2.

Процедура LZW-стиснення:
РЯДОК = черговий символ з вхідного потоку
WHILE вхідний потік не пустий DO
СИМВОЛ = черговий символ з вхідного потоку
IF РЯДОК + СИМВОЛ в таблиці рядків THEN
РЯДОК = РЯДОК + СИМВОЛ
ELSE
вивести в вихідний потік код для РЯДОК
добавити в таблицю рядків РЯДОК + СИМВОЛ
РЯДОК = СИМВОЛ
END of IF
END of WHILE
вивести в вихідний потік код для РЯДОК

Рисунок 2.2 - Алгоритм стиснення

Простий рядок, використаний для демонстрації алгоритму, наведено на рис. 2.3. Вхідний рядок є коротким списком англійських слів, розділених символом «/». Як ви можете помітити, аналізуючи алгоритм, його робота починається з того, що на першому кроці циклу він виконує перевірку на наявність рядка «/W» в таблиці. Коли він не знаходить цей рядок, то генерує код для «/» і додає в таблицю рядок «/W». Оскільки 256 символів вже визначені для кодів 0 – 255, то першому визначеному рядку

може бути поставлено у відповідність код 256. Після цього система читає наступну букву «E», добавляє другий підрядок «WE» в таблицю рядків і виводить код для букви «W». Коли знову буде прочитано рядок, що складається з символів «/» і «W», то йому буде поставлено у відповідність рядок з таблиці рядків. В цьому випадку система виводить код 256 і додає трисимвольний рядок в таблицю (/WE). Цей процес продовжується до тих пір, поки не закінчиться вхідний потік і всі коди не будуть виведені.

Вихідний потік для заданого рядка (/WED/WE/WEE/WEB/WET) показано на рис. 2.3, як і отриману таблицю рядків. Ця таблиця швидко заповнюється, оскільки новий рядок додається в таблицю кожний раз, коли генерується код. В цьому прикладі було виведено п`ять закодованих підрядків і сім символів. Звичайно, цей приклад було вибрано тільки для демонстрації. В дійсності стиснення не починається до тих пір, поки не буде збудована досить велика таблиця, зазвичай після читання порядка 100 вхідних байтів.

Вхідний рядок: /WED/WE/ WEE/ WEB/WET

Вхід (символи)	Вихід (коди)	Нові коди і відповідні рядки
/W	/	256 = /W
E	W	257 = WE
D	E	258 = ED
/	D	259 = D/
WE	256	260 = /WE
/	E	261 = E/
WEE	260	262 = /WEE
/W	261	263 = E/W
EB	257	264 = WEB
/	B	265 = B/
WET	260	266 = /WET
<EOF>	T	

Рисунок 2.3 – Процес стиснення

2.5.2 Відновлення інформації

Алгоритму стиснення відповідає свій алгоритм відновлення інформації. Він отримує вихідний потік кодів від алгоритму стиснення і використовує його для точного відновлення вхідного потоку. Однією з причин ефективності LZW - алгоритму є те, що він не потребує зберігання таблиці рядків, отриманої внаслідок стиснення. Таблиця може бути точно відновлена на основі вихідного потоку алгоритму стиснення. Це можливо тому, що алгоритм стиснення виводить РЯДКОВУ та СИМВОЛЬНУ компоненти коду раніше, ніж він помістить цей код у вихідний потік. Це означає, що стиснені дані не обтяжені необхідністю тягнути за собою велику таблицю рядків.

Алгоритм декодування даних, стиснутих методом LZW, наведено на рис. 2.4. Відповідно до алгоритму стиснення, він додає новий рядок в таблицю рядків кожний раз, коли читає з вхідного потоку новий код. Все, що йому необхідно зробити додатково - це перевести кожний вхідний код в рядок і переслати його в вихідний потік. Важливо відзначити, що побудова таблиці рядків алгоритмом декодування закінчується тоді, коли побудована таблиця рядків алгоритмом стиснення.

Процедура LZW - декодування:
читати СТАРИЙ_КОД
вивести СТАРИЙ_КОД
WHILE вхідний потік не пустий DO
читати НОВИЙ_КОД
РЯДОК = перевести НОВИЙ_КОД
вивести РЯДОК
СИМВОЛ = перший символ РЯДКА
добавити в таблицю рядків СТАРИЙ_КОД + СИМВОЛ
СТАРИЙ_КОД = НОВИЙ_КОД
END of WHILE

Рисунок 2.4 – Алгоритм декодування

Роботу цього алгоритму на основі стиснених даних, отриманих внаслідок стиснення розглянутого вище рядка

(/WED/WE/WEE/WEB/WET),

демонструє рис. 2.5.

Вихідний потік точно відповідає вхідному потоку алгоритму стиснення. Перші 256 кодів визначені для переводу одиничних символів, так само як і в алгоритмі стиснення.

Вхідні коди: / W E D 256 E 260 261 257 B 260 T

Вхід Новий код	**Старий код**	**Рядок Вихід**	**Символ**	**Новий вхід Таблиці**
/	/	/		
W	/	W	W	256 = /W
E	W	E	E	257 = WE
D	E	D	D	258 = ED
256	D	/W	/	259 = D/
E	256	E	E	260 = /WE
260	E	/WE	/	261 = E/
261	260	E/	E	262 = /WEE
257	261	WE	W	263 = E/W
B	257	B	B	264 = WEB
260	B	/WE	/	265 = B/
T	260	T	T	266 = /WET

Рисунок 2.5 – Процес декодування

2.6 Арифметичне кодування

Основні принципи арифметичного кодування були розроблені наприкінці 70-х років [24]. Арифметичне кодування, так само як і імовірностні методи, використовує як основу технології стиску імовірність появи символу у файлі, однак сам процес арифметичного кодування має принципові відмінності. Внаслідок арифметичного кодування символьна послідовність (рядок) замінюється дійсним числом більше нуля і менше одиниці.

Розглянемо процес арифметичного кодування слова «REDUNDANCE» (надмірність).

Імовірність появи кожного символу в цьому слові дорівнює 0.1 за винятком букв E, D і N, що зустрічаються двічі, і, відповідно, імовірність їхньої появи дорівнює 0.2. Далі кожній букві присвоюється інтервал імовірності (range), довжина якого розраховується, виходячи з імовірності їхньої появи в слові (табл. 2.2).

Перша буква слова – «R» – одержує інтервал з нижньою границею 0.8 і з верхньою – 0.9. Нижня границя інтервалу і стає першою значущою цифрою коду. Потім виконується розрахунок границь підінтервалів для кожної наступної букви за такими виразами:

$$\beta_n^l = \beta_{n-1}^l + (\beta_{n-1}^h - \beta_{n-1}^l)P_n^l$$
$$\beta_n^h = \beta_{n-1}^l + (\beta_{n-1}^h - \beta_{n-1}^l)P_n^h,$$

де β^l, β^h – нижня і верхня границя кодового інтервалу,
P^l і P^h – нижня і верхня границі інтервалу імовірності для символа.

Результат – послідовність символів 'REDUNDANCE' заміняється числом 0.8478570048, що подано в табл. 2.3. Таким чином, замість 10 байт, необхідних для збереження символьного рядка, буде потрібно всього 4 байти для запису числа.

Арифметичне кодування дозволяє забезпечити високий ступінь стиску даних, особливо у випадках, коли зустрічаються дані, де частота появи різних символів сильно відрізняється одна від одної.

У той самий час сама процедура арифметичного кодування потребує потужних обчислювальних ресурсів, і донедавна цей метод мало застосовувався під час кодування зображень через повільну роботу алгоритму і, відповідно, істотного часу затримки в процесі передачі даних. Хоча, варто очікувати високих коефіцієнтів стиску за його застосування для кодування зображень.

Таблиця 2.2 – Інтервали імовірності для символів в слові Redundance

Символ	Імовірність (p)	Інтервал
A	0.1	0.0-0.1
C	0.1	0.1-0.2
D	0.2	0.2-0.4
E	0.2	0.4-0.6
N	0.2	0.6-0.8
R	0.1	0.8-0.9
U	0.1	0.9-1.0

Таблиця 2.3 – Покрокове подання рядка REDUNDANCE методом арифметичного кодування

Символ	Нижня границя (β^l)	Верхня границя (β^h)
R	0.8	0.9
E	0.84	0.86
D	0.844	0.848
U	0.8476	0.848
N	0.84784	0.84792
D	0.847856	0.847872
A	0.8478560	0.8478576
N	0.84785696	0.84785728
C	0.847856992	0.847857024
E	0.8478570048	0.8478570432

2.7 Шифрування і стиснення зображень

Шифрування і стиснення даних - надзвичайно пов’язані задачі. В обох випадках змінюється частотний розподіл символів у вихідних даних. Зменшення надлишковості (стиснення) початкових даних значно підвищує криптостійкість алгоритмів шифрування.

Наукова криптологія (cryptos – таємний, logos – слово) бере початок з роботи К.Шеннона “Теорія зв’язку в секретних системах” (1949р.), в якій було показано, що для деякого випадкового шифру кількість знаків шифротексту, отримавши який криптоаналітик за необхідних обчислювальних ресурсів зможе відновити ключ (тобто розкрити шифр), становить:

$$n = \frac{H(Z)}{r \log N}, \qquad (2.1)$$

де *H(Z)* – ентропія ключа,

r – надлишковість відкритого тексту,

N – обсяг алфавіту.

З виразу (2.1) видно, що зниження надлишковості (стиснення даних) може значно збільшити криптостійкість навіть для коротких ключів [22].

2.7.1 Шифрування інформації

Найбільш простими в реалізації є методи прямої підстановки та перестановки. В прямих підстановках кожний знак початкового тексту замінюється одним або декількома іншими знаками. Розрізняють:

- моноалфавітну підстановку
- багатоалфавітну підстановку.

За моноалфавітної підстановки встановлюється відповідність між кожним знаком a_i алфавіту повідомлення A і відповідного знаку зашифрованого тексту. Всі методи моноалфавітної підстановки можливо подати як числові перетворення букв початкового тексту відповідно до виразу:

$$C = (ap + S) \bmod k, \qquad (2.2)$$

де a – десятковий коефіцієнт,

S – коефіцієнт зсуву,

k – розмір алфавіту,

p – символ початкового тексту [23]

Недоліком моноалфавітної підстановки є низька криптостійкість, оскільки в шифрованому повідомленні зберігається частотний розподіл символів початкового тексту.

Ці недоліки можна зменшити за рахунок використання багатоалфавітної підстановки. За n-алфавітної підстановки знак m_1 з початкового повідомлення замінюється знаком з алфавіту B_1, m_2 – відповідно з алфавіту B_2, ..., m_n – знаком з алфавіту B_n, m_{n+1} – знову з алфавіту B_1 і т. д. Ефект використання багатоалфавітної підстановки полягає в тому, що забезпечується маскування природної частотної статистики початкової мови повідомлення, оскільки конкретний знак з алфавіту A може бути перетворений в декілька різних знаків шифрувального алфавіту B.

Знаки початкового тексту можна також переставляти згідно з деяким правилом. Недолік перестановок той самий, що і моноалфавітної підстановки.

Ефективність шифрування можливо підвищити, використовуючи комбінацію перестановок і підстановок.

Значно більшу криптостійкість забезпечує шифрування з використанням генератора псевдовипадкових чисел (ПВЧ). Найбільш широке застосування знаходять лінійні конгруентні генератори ПВЧ. Цей генератор формує послідовність псевдовипадкових чисел T_1, T_2,..., T_m, використовуючи співвідношення:

$$T_{i+1} = (aT_i + c) \bmod m, \qquad (2.3)$$

де a і c – константи,

T_0 – початкова величина, вибрана як породжувальне число.

Період повторення псевдовипадкових чисел залежить від вибраних значень a і c. Лінійний конгруентний генератор має максимальну довжину m тільки тоді, поки c – непарне, а $a \bmod 4 = 1$.

Під час шифрування псевдовипадкові числа, отримані з виразу (2.3), об'єднуються деяким чином з відповідним обсягом тексту, але так, щоб можна було відновити початковий текст.

Наприклад, з використанням додавання за модулем 2 (накладання гами-ключа на початковий текст). Якщо періодичність генератора більша довжини всіх посланих повідомлень початкового тексту і якщо криптоаналітику невідомий ніякий початковий текст, то шифр теоретично неможливо відкрити [24].

Такі відомі схеми кодування як DES, RSA, алгоритм криптографічного перетворення згідно з ГОСТ 28147-89 повністю або частково є комбінацією методів, розглянутих вище.

2.7.2 Одночасне стиснення і шифрування даних

Основною метою кодування або стиснення даних є перетворення вхідного потоку символів в потік бітів мінімальної довжини. Це досягається за рахунок зменшення надлишковості вихідного потоку. У цьому випадку символи, які найбільш часто зустрічаються, подаються короткими кодовими комбінаціями, за рахунок чого і досягається стиснення. Однак, під час стиснення даних деякими методами існує можливість одночасного шифрування або без додаткових обчислювальних затрат, або з низькими затратами. Оскільки в більшості випадків, файли, які передаються по комп'ютерних мережах або зберігаються в пам'яті, подано в стиснутому вигляді, то було б доцільно надати можливість користувачам виконувати під час стиснення файлів і їх шифрування.

Розглянемо з цього погляду деякі методи стиснення інформації.

Достатньо простий і ефективний метод стиснення даних з невідомим розподілом частот, відомий як кодування за ступенем новизни (див. п. 2.4).

Під час стиснення інформації цим методом можливе одночасне виконання шифрування методом багатоалфавітної підстановки без додаткових затрат, або з застосуванням інших методів і їх комбінацій з незначними додатковими затратами.

Нехай передається повідомлення:

$$\omega = \mathbf{ABCDDAAACAB}$$

в алфавіті **AL={A,B,C,D}**. За появи в слові ω чергової букви «а» по лінії зв'язку передається монотонний код номера позиції, яку займає в цей момент символ «а» в списку, а символ «а» переміщується на початок списку.

Таким чином символи, які часто зустрічаються, завжди будуть знаходитись на початку списку і відповідно подаватись короткими

кодовими комбінаціями. Порядок кодування за ступенем новизни демонструє табл. 2.4.

Таблиця 2.4 – Кодування за ступенем новизни

Вхід	Вихід1	Алфавіт1	Вихід2	Алфавіт2
A	0_2	{ABCD}	110_2	{CBAD}
B	10_2	{ABCD}	110_2	{ACBD}
C	110_2	{BACD}	110_2	{BACD}
D	111_2	{CBAD}	111_2	{CBAD}
D	0_2	{DCBA}	0_2	{DCBA}
A	111_2	{DCBA}	111_2	{DCBA}
A	0_2	{ADCB}	0_2	{ADCB}

Якщо список алфавіту «AL» згенерувати, наприклад, використовуючи генератор ПВЧ, то вихідна послідовність стане іншою (табл. 2.4 – вихід 2, алфавіт2). Не знаючи початкового стану алфавіту, неможливо дешифрувати повідомлення. У цьому разі реалізується шифрування методом багатоалфавітної підстановки і стиснення даних одночасно. Для одночасного стиснення і шифрування даних методом багатоалфавітної підстановки може використовуватись така схема:

- згідно з формулою (2.3) генерується алфавіт повідомлення. Оскільки символи в комп'ютерних системах подано 8-бітовими комбінаціями, то m = 256, а константи a, c і породжувальне число T_0 можна вводити як ключ шифру;
- виконується стиснення згідно з наведеним методом.

Додаткові затрати відсутні, оскільки генерація символів алфавіту виконується в будь-якому випадку.

За незначних додаткових затрат можна застосувати комбінацію методів, яка містить два основні етапи:

- Етап 1. Кодування за ступенем новизни для стиснення і шифрування методом багатоалфавітної підстановки.
- Етап 2. Шифрування за допомогою додаткового генератора ПВЧ з m, що дорівнює довжині файлу, який отримано як результат виконання етапу 1. Цей або інший генератор ПВЧ може бути використаний також для виконання перестановок, оскільки він генерує всі числа в межах довжини початкового файлу. Якщо в стисненні даних немає потреби, то ця перестановка дозволить замаскувати стиснені дані в межах файлу такого самого розміру як і початковий, з попередньо записаною випадковою інформацією.

Аналіз таких відомих алгоритмів стиснення інформації як кодування Гаффмана та LZW показує також, що їх легко адаптувати до одночасного виконання стиснення і шифрування інформації. А за деяких додаткових обчислень згідно з формулою (2.3), ще і шифрування методом ПВЧ. Для цього достатньо лише сформувати таблицю перекодування кожного зчитаного з пам'яті символу. Ця таблиця може бути сформована, наприклад, за допомогою генератора ПВЧ (2.3) з $m = 256$. У цьому випадку зчитаний з початкового файлу символ даних задає позицію в таблиці перекодування, з якої вибирається код підстановки.

2.8 Запитання для самоконтролю

1. Наведіть класифікацію алгоритмів стиснення без втрат.
2. Охарактеризуйте словарні методи стиснення зображень.
3. Охарактеризуйте статистичні методи стиснення зображень.
4. Дайте характеристику стиснення зображень методом RLE.
5. Які недоліки кодування Хаффмена?
6. Наведіть порядок побудови кодового дерева.
7. Що таке префіксні коди?
8. Наведіть алгоритм кодування зображень методом LZW.
9. Наведіть алгоритм декодування зображень методом LZW.
10. Поясніть основи арифметичного кодування.
11. Які недоліки алгоритму LZW?
12. Чи забезпечують алгоритми стиснення без втрат високий коефіцієнт стиснення зображень?
13. Який основний недолік арифметичних методів стиснення зображень?
14. На якому етапі кодування зображень застосовуються алгоритми стиснення без втрат?
15. Як пов'язані шифрування і стиснення даних?

3 JPEG, MPEG ТА ФРАКТАЛЬНИЙ МЕТОД

3.1 Кодування зображень відповідно до стандартів JPEG та MPEG

Досягти великих коефіцієнтів стиснення, використовуючи один метод (крім фрактального), практично неможливо. Тому ефективне кодування зображень виконується із застосуванням декількох методів за декілька етапів. Цю концепцію і покладено в основу стандартів, розроблених міжнародною організацією зі стандартизації (ISO), які відомі як стандарти JPEG (Joint Photographic Expert Group) та MPEG (Motion Picture Experts Group). Стандарт JPEG призначений для стиснення нерухомих зображень, а MPEG – рухомих зображень [25-26].

Процес кодування за схемою JPEG поділяється на такі етапи (рис. 3.1):

1. Перетворення зображення в оптимальний кольоровий простір (тільки у випадку кодування кольорових зображень).
2. Субдискретизація компонент різницевих кольорових сигналів шляхом усереднення груп пікселів (тільки у випадку кодування кольорових зображень).
3. Виконання ДКП для зменшення надлишковості даних зображення.
4. Квантування кожного блока коефіцієнтів ДКП із застосуванням вагових функцій, оптимізованих з урахуванням сприйняття людиною.
5. Кодування результувальних коефіцієнтів із застосуванням статистичного кодування Гаффмана.

Рисунок 3.1 – Послідовність операцій в процесі стиснення зображень за методом JPEG

Під час виконання першого етапу виконується перетворення кольорового зображення з системи подання RGB в іншу систему, наприклад *YUV*, де *Y* – сигнал яскравості, *U* і *V* – різницеві кольорові 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* – кольорові сигнали зображення (червоний, зелений і синій відповідно).

Подання кольорового зображення в системі *YUV* дає можливість використати особливості зорового сприйняття зображень людиною – низьку чутливість до точності подання кольорів. Це виконується за рахунок субдискретизації компонент різницевих кольорових 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 2×2, зменшує розмір компонент *U* і *V* в чотири рази без помітного зменшення якості зображення. Враховуючи те, що на кожний піксел витрачається три байти (*YUV*), виграш значний.

Найбільш важким для реалізації є виконання ДКП. Однак, завдяки наявності швидких алгоритмів (ті самі, що для обчислення ДПФ) число арифметичних операцій може бути зменшено в десятки разів. Процес кодування із застосуванням ДКП пояснює рис. 3.2.

Рисунок 3.2 - Кодування з застосуванням ДКП

Зображення розбивається на примикальні один до одного блоки розміром 8×8 (під час кодування кольорових зображень кожна компонента обробляється незалежно). В межах кожного блока виконується двовимірне ДКП згідно з виразом:

$$F(u,v) = \frac{1}{4} C(u) C(v) \sum_{I=0}^{7} \sum_{J=0}^{7} f(i,j) \cos\left(\frac{(2i+1)u\pi}{16}\right) \cos\left(\frac{(2j+1)v\pi}{16}\right), \quad (3.1)$$

де

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

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

Під час декодування обчислюється зворотне ДКП:

$$f(i,j) = \frac{1}{4} \sum_{U=0}^{7} \sum_{V=0}^{7} C(u)C(v)F(u,v)\cos(\frac{(2i+1)u\pi}{16})\cos(\frac{(2j+1)v\pi}{16}), \qquad (3.2)$$

де $i, j = 0, 1, 2 \ldots 7$.

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 заданих яскравості і кольору застосовуються різні таблиці квантування, оскільки око людини має різну чутливість до яскравості і кольору зображення.

На етапі статистичного кодування специфікація JPEG допускає застосування крім алгоритму Гаффмана і інших методів з метою зменшення обсягу інформації.

Під час кодування рухомих зображень відповідно до стандарту MPEG застосовуються два типи стиснення: внутрікадрове кодування (подібно JPEG) і міжкадрове кодування. Міжкадрове кодування основано на кодуванні з передбаченням та інтерполяцією. Кадри рухомого зображення містять багато ідентичних даних. Якщо використати цей факт, то як результат – набагато збільшиться ступінь стиснення.

Для підтримки міжкадрового і внутрікадрового кодування потік даних MPEG містить три типи закодованих кадрів:

- I-кадри,
- P-кадри,
- B-кадри.

I-кадр містить один кадр відеоданих, який не пов'язаний з інформацією у будь-якому іншому кадрі. P-кадр містить відмінності між поточним та попереднім I- або P-кадром. B-кадр містить відмінності між поточним кадром і двома (попереднім і наступним) I- або P-кадрами. Типова послідовність кадрів в потоці MPEG виглядає приблизно так:

IBBPBBPBBPBBIBBP...(0123456789...).

Декодовані вони будуть в послідовності:

IPBBPBBPBB...(0312645978...).

А відображені в послідовності:

IBBPBBPBBP...(0123456789...).

I-, P-, B-кадри стискаються з використанням ДКП і статистичного кодування Гаффмана подібно JPEG.

Перша модифікація стандарту MPEG (MPEG-1) орієнтована на застосування в комп'ютерних системах для відтворення кольорових рухомих зображень в форматі 352×240 з частотою 30 кадрів за секунду. Наступні модифікації стандарту MPEG (MPEG-2, MPEG-4) розраховані на більшу роздільну здатність і можуть знайти застосування в цифровому телебаченні. Серед недоліків JPEG і MPEG кодування необхідно відзначити такі як:

1. Недостатні коефіцієнти стиснення складних зображень.
2. Неможливість зміни роздільної здатності.
3. Деяка втрата достовірності відновлених зображень.

Однак, незважаючи на зазначене, ці методи поки що мають найкращі реально досягнуті характеристики за сукупністю таких параметрів як коефіцієнт стиснення, якість, швидкодія і підтримуються основними виробниками комп'ютерної техніки та техніки зв'язку. Додаткову інформацію можна знайти в Internet, наприклад, за адресою http://www.mpeg.org.

3.2 Фрактальне стиснення зображень

З відомих методів кодування зображень фрактальний метод дозволяє отримувати найбільші коефіцієнти стиснення. З фізичного погляду фрактальне кодування ґрунтується на твердженні, що зображення містить афінну надлишковість [5]. Математична модель, яка використовується за фрактального стиснення зображень, називається системою ітерованих функцій (Iterated Function System – IFS). Система ітерованих функцій містить набір стискальних перетворень w_i, які можливо задати так:

$$W(\cdot) = \bigcup_{i=1}^{n} w_i(\cdot) \qquad (3.3)$$

Хатчинсон [2] показав, що для вхідного зображення f_0 результат перетворення

$$f_\infty = \lim_{n\to\infty} W^{0n}(f_0) \qquad (3.4)$$

не залежить від вибору f_0. Зображення f_∞ називається нерухомою точкою перетворення **W**. Як перетворення w_i використовуються афінні перетворення:

$$w_i \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & S_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} dx \\ dy \\ O_i \end{bmatrix} \quad (3.5)$$

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, ми можемо відновити початкове зображення.

3.2.1 Алгоритм кодування-декодування зображень фрактальним методом

Одна з можливих схем кодування зображень фрактальним методом запропонована Жакеном (Jacquin) [15,18], і містить такі етапи:

- Зображення розділяється на області, що примикають одна до одної розміром $n \times n$ (рангові області).
- Задається набір доменних областей. Доменні області можуть перекриватись, вони не мають обов'язково закривати всю поверхню зображення. Розміри доменних областей звичайно вибирають $2n \times 2n$.
- 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 наведено на рис. 3.3 [15 – 19]. 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 до його сторін.
- Точність апроксимації визначається за допомогою середньо-квадратичного критерію:

$$F = \sum_{i,j} (S d_{ij} + O_{ij} - r_{ij})^2 \quad (3.6)$$

де d_{ij} – значення, отримані внаслідок усереднення по фрагментах 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 (як в цій роботі), так і описуватись поліномами першого, другого, третього порядків [3].

Прирівнявши до нуля часткові похідні від виразу (3.6) по S і O:

$$\frac{dF}{dS} = 0, \qquad \frac{dF}{dO} = 0,$$

знайдемо значення S і O, за яких досягається мінімум виразу (3.6):

$$O = \frac{1}{n^2}\left(\sum_{i,j}^{n} r_{ij} - S\sum_{i,j}^{n} d_{ij}\right) \tag{3.7}$$

$$S = \frac{n^2 \sum_{i,j}^{n} r_{ij} d_{ij} - \sum_{i,j}^{n} r_{ij} \sum_{i,j}^{n} d_{ij}}{n^2 \sum_{i=j}^{n} d_{ij}^{\,2} - (\sum_{i,j}^{n} d_{ij})^2} \tag{3.8}$$

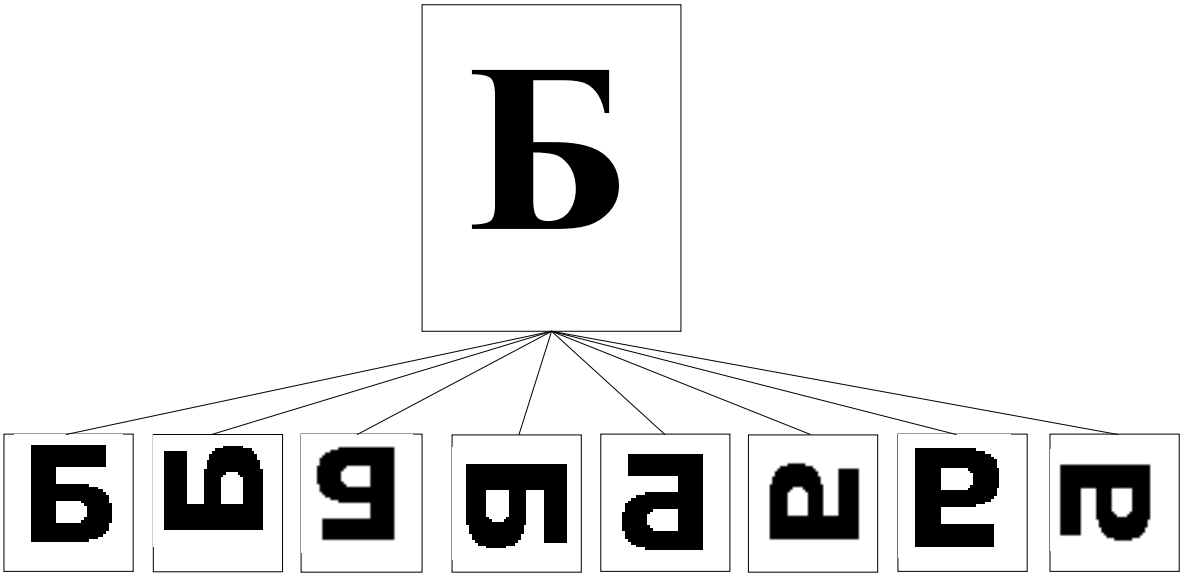


Рисунок 3.3 – Можливі варіанти відображення одного квадрата в інший з використанням афінних перетворень

Доменні блоки звичайно вибирають з кроком *n*/2 для *n*=4. У вихідний файл записуються такі параметри:

- координати доменної області з найменшим значенням *Fmin*;
- значення для *O* і *S*, отримані згідно з формулами (3.7, 3.8);
- номер афінного перетворення.

Процес фрактального перетворення асиметричний. Тобто, процес декодування не є простою інверсією процедур стиснення і потребує значно менше часу для його виконання, що дає можливість уже тепер використовувати цей метод для зберігання зображень на компакт-дисках та інших носіях інформації. Декодування стисненого зображення носить ітераційний характер і складається з таких етапів.

Створюються два зображення однакового розміру А і Б. Розмір цих зображень не обов’язково дорівнює розміру початкового зображення, початковий рисунок областей А і Б – 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 нерозрізненними.

Основним недоліком фрактального методу є низька швидкість кодування, яка пов'язана з тим, що для отримання високої якості зображення для кожного рангового блока необхідно виконати перебір всіх доменних блоків, і для кожного доменного блока 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 під час кодування зображень фрактальним методом (вважаючи, що S=1), оскільки для їх виконання витрачається найбільше часу. Нехай $M \times N$ – розміри зображення і $M=N=n^k$, де n – розмір сторін рангового блока. Нехай крок вибору доменних блоків $n/2$. Для порівняння цього рангового блока з доменним блоком необхідно виконати $8n^2$ операцій множення. Тоді загальна кількість операцій множення для пошуку відповідного доменного блока така:

$$M = 8n^2 * (\text{кількість доменних блоків}),$$

Кількість доменних блоків за кроку вибору $n/2$ становитиме:

$$(\frac{n^k - 2n}{n/2} + 1) \cdot (\frac{n^k - 2n}{n/2} + 1).$$

Після підстановки кінцева формула має такий вигляд:

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

Для порівняння, виконання ДКП для блока такого ж розміру, навіть без використання швидких алгоритмів, потребує лише n^4 операцій множення.

Незважаючи на те, що виконані в останні роки роботи дозволили зменшити час кодування в десятки разів [15, 19], актуальним залишається проведення досліджень з підвищення швидкості стиснення зображень цим методом в таких напрямках:

- пошук критеріїв, які дозволяли б швидко виконувати відбір доменних блоків без перебору всіх афінних перетворень;
- зменшення числа афінних перетворень.

3.2.2 Підвищення швидкодії фрактального стиснення

Найпростіший і найповільніший спосіб фрактального кодування є перевірка кожного доменного блока і виконання обчислень згідно з формулами (3.6 – 3.8). Такий спосіб називається **повним пошуком або повним перебором.** Під час кодування зображень природного походження можна підвищити швидкодію кодування, взявши S=1, оскільки, враховуючи статистику зображень, завжди знайдеться доменний блок, який апроксимує заданий ранговий блок з необхідною точністю. Тоді з виразів (3.6, 3.7) одержимо:

$$F = \sum_{i,j} (d_{ij} + O_{ij} - r_{ij})^2 \tag{3.9}$$

$$O = \frac{1}{n^2}\left(\sum_{i,j}^{n} r_{ij} - \sum_{i,j}^{n} d_{ij} \right) \tag{3.10}$$

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 того погляду, що вже в самому алгоритмі передбачена візуально недостовірна передача деяких графічних зображень штучного походження, тому необхідні додаткові заходи для забезпечення хоча б візуальної достовірності.

Серед відомих способів підвищення швидкодії кодування зображень фрактальним методом можна виділити такі [15, 18, 19]:

- пошук доменних блоків, для яких *F* не перевищує заданого значення;
- локальний та сублокальний пошук;
- ізометричне передбачення;
- класифікація доменних і рангових блоків. Ранговий блок порівнюється з доменними блоками того ж самого класу.

Кодування відповідно першого підходу пояснює рис. 3.4.

```
* Вибрати прийнятний рівень Ec і установити розмір рангових
  блоків  Ri  n=nmax.
* Помітити блоки Ri як непокриті;
* Поки є непокриті Ri  {
   * Знайти Di і відповідне перетворення wi яке дає мінімум F;
   *     Якщо F< Ec або розмір Ri n ≤ nmax, то
         * Відмітити Ri як покритий і записати параметри wi;
   * Інакше
         * Розділити Ri на менші блоки і помітити їх як непокриті
}
```

Рисунок 3.4 – Пошук доменних блоків з заданим *F*

Тобто, передбачається змінний розмір доменних і рангових блоків. За постійного розміру доменних і рангових блоків n = const пошук продовжується до тих пір, поки не буде знайдений доменний блок, який забезпечує $F < E_c$ [18].

Локальний пошук (Local Search) передбачає пошук доменного блока тільки на ділянках зображення сусідніх з цим ранговим блоком.

Збільшити швидкість кодування можливо також, збільшивши крок вибору доменних блоків, що 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 блока з меншим кроком (рис. 3.5). Такий спосіб носить назву **локальний субпошук** (Local Sub Search – LSS).

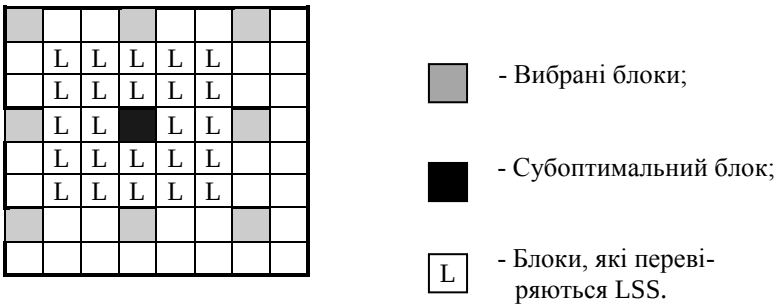


Рисунок 3.5 – Локальний субпошук

Кількість афінних перетворень зменшується за рахунок **ізометричного передбачення.** Кожний доменний (ранговий) блок розділяється на чотири субблоки, обчислюється середнє значення яскравості в межах кожного субблока і виконується сортування цих субблоків в порядку зростання. Порівнюючи сортований і несортований списки, обчислюється лексикографічний порядок перестановок [29]:

$$\text{Perm \#} = \text{inv}(x_1)\cdot 3! + \text{inv}(x_2)\cdot 2! + \text{inv}(x_3)\cdot 1! + \text{inv}(x_4)\cdot 0!, \qquad (3.11)$$

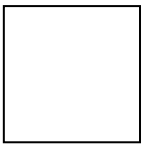
де $\text{inv}(x_i)$ – кількість перестановок, яка відноситься до *i*-ої позиції (рис. 3.6).

Тепер кожний блок має свій лексикографічний порядок, що дає можливість визначити найкраще покриття рангового блока.

Серед методів з класифікацією необхідно відзначити класифікацію, запропоновану Жакеном [19], яка грунтується на топології блоків і передбачає блоки трьох типів:

- блоки без контурів (shade blocks);
- блоки, інваріантні до орієнтації (текстурні блоки);
- контурні блоки (edge blocks).

Shade-блоки можуть бути апроксимовані за допомогою поліномів третього або другого порядку, і пошук для них непотрібен, пошук текстурних блоків виконується без перевірки ізометрії (без виконання афінних перетворень). Тільки для контурних блоків виконується повний перебір.



2	3
0	1

[0,1,2,3]

L (#) - середня яскравість субблока

L (2) > L (0) > L (1) > L (3)

[0,1,2,3] → [2,0,1,3]

Perm # = 2· 3! + 1· 2! + 1· 1! + 0· 0! = 15.

Рисунок 3.6 – Приклад обчислення лексикографічного порядку

3.2.3 Табличний метод підвищення швидкодії

Обсяг пам'яті та швидкодія комп'ютерів кожний рік подвоюються. Це дозволяє очікувати, що стане можливим застосування табличних методів пошуку доменних блоків без попередньої обробки зображень і пошуку критеріїв класифікації доменних блоків.

Доменний блок можна подати як вектор з n^2 координатами

$$\vec{D}(d_1, d_2, ..., d_{n^2}),$$

де d_i – значення яскравості елементів блока,

n – розмір сторін рангового блока.

Після виконання афінних перетворень утворюються ще 7 векторів, координати яких є перестановками координат початкового вектора. Якщо для подання елемента d_i витрачається 8 біт, то розрядність числа $d_1 d_2 ... d_{n^2}$ буде дорівнювати 2^{8n^2}.

Формуються таблиці, подані на рис. 3.7. Як видно з рисунка, для значення координати рангового блока «k» вибираються доменні блоки з номерами m та $m1$, для яких необхідно обчислити тільки значення F, S та O згідно з виразами (3.6 – 3.8).

Зрозуміло, що розміри таблиці 1 на рис. 3.7 не дозволяють поки що реалізувати такий підхід на практиці без попередньої обробки доменних блоків для скорочення розмірності подання елементів цих блоків.

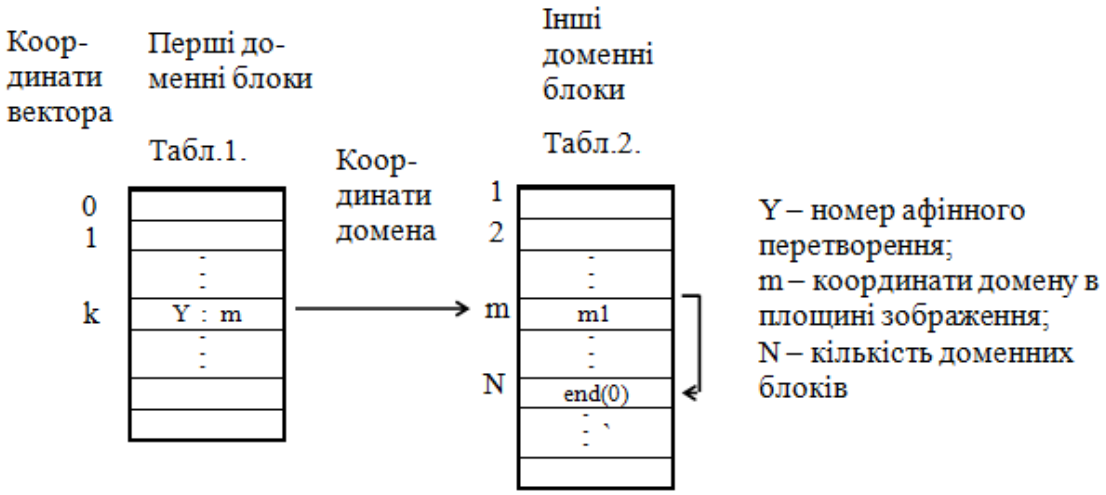


Рисунок 3.7 – Табличний метод пошуку

Підвищення швидкодії фрактального стиснення ґрунтується на попередній обробці кожного доменного і рангового блока одним з операторів виділення контурів зображення і оціненні детальності кожного з цих блоків [5, 6]. Найбільші переваги має використання фільтра Лапласа, оскільки він точно ідентифікує контури і порівняно градієнтними операторами забезпечує менші розміри таблиць пошуку:

$$G(i,j)= 4\,f(i, j) - f(i\text{-}1, j) - f(i\text{+}1, j) - f(i, j\text{-}1) - f(i, j\text{+}1), \qquad (3.12)$$

де $f(i,j)$ – значення відліків вхідного зображення.

Обробка блоків згідно з виразом (3.12) виключає з порівняння постійну складову зображення і тому дозволяє попередньо відібрати близькі до цього рангового блока доменні блоки.

Під час першого проходу оцінюється детальність доменних і рангових блоків. Такою оцінкою може бути сума модулів значень відліків контурної компоненти:

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

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

де k_R – номер рангового блока,

k_D – номер доменного блока,

S_{k_D}, S_{k_R} – детальність відповідно доменного та рангового блока.

Суми S_{k_R} заносяться в масив, в порядку слідування рангових блоків, а для доменних блоків формуються таблиці, наведені на рис. 3.8.

Під час другого проходу для кожного значення детальності рангового блока S_{k_R} з таблиць 1, 2 (рис. 3.8) відбираються відповідні доменні блоки, для яких виконується обробка, характерна для фрактального стиснення та обчислення згідно з виразами (3.6 – 3.8). Оскільки вибраних блоків значно

менше загальної кількості доменних блоків, то потрібно очікувати значного виграшу в швидкодії.

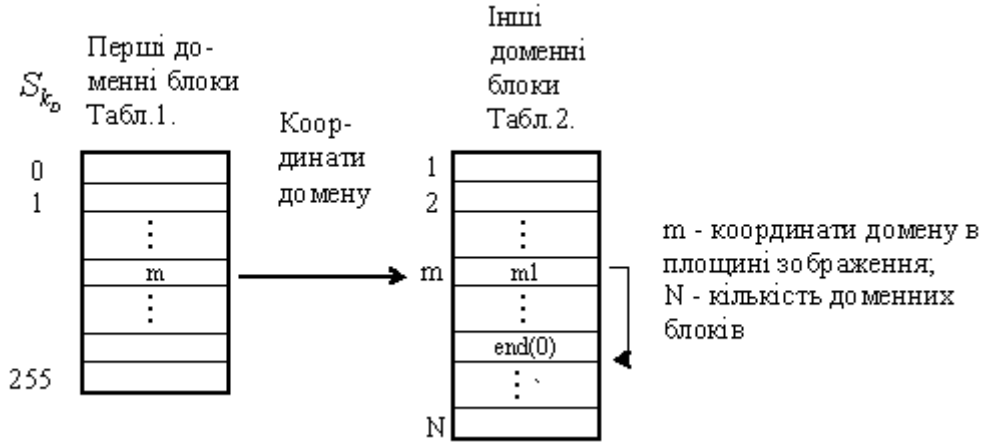


Рисунок 3.8 – Пошук доменного блока згідно з критерієм детальності

Приклад декодування зображення фрактальним методом наведено на рис. 3.9.

Початкове зображення

Декодоване зображення - три ітерації

Декодоване зображення - 8 ітерацій

Декодоване зображення - 16 ітерацій

Рисунок 3.9 – Декодування зображення фрактальним методом

3.3 Запитання для самоконтролю

1. Наведіть етапи кодування зображень згідно з стандартом JPEG.
2. Чому виконується перетворення із системи RGB в YUV?
3. Наведіть математичні вирази для прямого і зворотного ДКП.
4. Чому зображення розбивається на квадрати 8×8?
5. Які відмінності між MPEG і JPEG?
6. Які зображення стискає метод JPEG?
7. Які зображення стискає метод MPEG?
8. Які методи застосовуються у разі стиснення рухомих зображень?
9. Які недоліки MPEG і JPEG?
10. Наведіть алгоритм стиснення зображень фрактальним методом?
11. Який основний недолік фрактального методу?
12. Наведіть алгоритм декодування зображень фрактальним методом.
13. Охарактеризуйте основні методи підвищення швидкодії фрактального стиснення зображень.
14. Охарактеризуйте Wavelet-кодування.
15. Наведіть алгоритм кодування зображень методом Wavelet.
16. Які недоліки Wavelet-кодування?
17. Охарактеризуйте перспективи розвитку кодування зображень.

4 РЕКУРСИВНИЙ АЛГОРИТМ ТА JPEG 2000

Останнім часом інформаційні технології набули особливого статусу. Якщо ще кілька десятиліть тому мало хто мав хоч якесь уявлення про цю нову і досить прогресивну галузь знань, сьогодні нелегко знайти людину, яка так чи інакше не стикалася з її практичними додатками. Повсюдна комп'ютеризація та інформатизація, цифровий звук, цифрове відео, цифрова телефонія – це лише окремі приклади впровадження інформаційних технологій у побуті. За прогнозами спеціалістів, найближчими роками частка подібних додатків лише збільшуватиметься. У зв'язку з цим, подальше вивчення прикладних аспектів теорії інформації набуває особливого значення. Деякі з технологій, що розробляються, вже сьогодні виглядають багатообіцяюче. Ймовірно, одним із найяскравіших прикладів цього можуть бути методи цифрової обробки зображень.

Проблема обробки відеосигналу аж ніяк не нова. Вже починаючи з середини XX століття, вчені займаються активними дослідженнями у цій галузі. Наслідком їхніх зусиль на початковому етапі стала поява аналогового телебачення. Формування, передача і відображення картинки, що рухається, виявляються можливими завдяки застосуванню нескладних аналогових елементів. Практичний ефект від подібної новинки відразу привертає увагу фахівців, що приводить до справжнього телевізійного та кіноіндустріального буму. Ближче до кінця XX століття дослідники все частіше спрямовують свої зусилля на розробку цифрових технологій передачі та обробки сигналу. Перевага цифрового підходу над аналоговим очевидна: цифрове подання зручніше у використанні, воно дозволяє досягти більшої якості і часом більшої місткості. Наприкінці восьмидесятих перспективність використання цифрового подання вже не викликає сумнівів, а додатковий поштовх до розвитку цифрових методів дає поява персональних комп'ютерів.

Перші персональні обчислювальні машини були малопродуктивні до виконання складних завдань з обробкою потокового відео. Проте дуже часто продуктивності вистачало для здійснення маніпуляцій з окремими зображеннями. У зв’язку з цим відбулося виділення двох складових проблеми обробки відео. Проблема обробки потоку зображень і проблема обробки поодиноких зображень тепер були двома окремими завданнями, причому, маючи у своєму розпорядженні не досить швидкі, але, що важливо, недорогі персональні обчислювальні машини, дослідники мали значно більше можливостей для вирішення саме останнього завдання. Як наслідок, саме обробка нерухомих зображень стала одним із основних додатків персональних комп'ютерів.

Під обробкою нерухомих зображень зазвичай маються на увазі не лише маніпуляції, що приводять до їх зміни, але також і операції, що дозволяють змінювати фізичне подання зображень без візуально відчутних наслідків. Останнє прийнято називати ущільненням зображень.

Процес ущільнення приводить до зменшення обсягу подання інформації на інформаційному носії. Існує два типи ущільнення: ущільнення, пов'язане зі спотвореннями подання інформації; ущільнення, що не допускає таких спотворень. Перше має назву ущільнення з втратами, а друге – ущільнення без втрат. Із зрозумілих причин ущільнення з втратами застосовується далеко не до всіх типів інформації, проте там, де воно все ж таки може бути використане, воно, як правило, виявляється значно ефективнішим порівняно з ущільненням без втрат. У випадку з нерухомими зображеннями ущільнення з втратами виявилося не тільки застосовним, а й більш ніж виправданим. Як виявилося, людське сприйняття не вловлює деякі візуальні деталі; втрата чи спотворення цих деталей відповідно є абсолютно безболісними для людини.

Дослідниками було розроблено величезну кількість методів ущільнення нерухомих зображень, але це не гарантує можливість коректної інтерпретації подання особою, яка не залучена до процесу кодування. Таким чином, постає проблема уніфікації методів, тобто проблема стандартизації. Координацією зусиль стандартизації методів ущільнення зображень традиційно займається міжнародна організація стандартизації (ISO) в особі спеціальної групи експертів з фотографічних зображень (JPEG), а також відділення стандартизації міжнародного телекомунікаційного союзу (ITU-T) [1].

Розробка першого повноцінного стандарту, розпочата 1986 року, увінчалася появою 1992 року міжнародного стандарту JPEG [1]. Формування стандарту проводили на конкурсній основі. Серед безлічі пропозицій було вибрано одну, яка була взята за основу стандарту; надалі пропозиція була вдосконалена зусиллями різних спеціалістів. JPEG загалом виявився досить вдалим стандартом. Він забезпечує високу ефективність ущільнення за прийнятного рівня втрат і допускає ефективні практичні реалізації. Стандарт JPEG широко використовується практично з моменту самої появи. Він став одним із найпопулярніших стандартів зберігання інформації. На сьогодні стандарт підтримує практично кожен додаток, орієнтований на обробку відеозображень. Незважаючи на величезну популярність стандарту JPEG, через кілька років природним чином виникла необхідність у появі нового стандарту. Були розроблені ефективніші методи ущільнення, а програми потребували від форматів зберігання відеоданих все більше і більше функціональності. Стандарт JPEG часто не задовольняв нові вимоги.

Назва нового стандарту – JPEG-2000 – спочатку вказувала на рік очікуваного виходу. На жаль, на нині цей стандарт так і не набув популярності стандарту JPEG, хоча основні етапи його затвердження успішно пройдені, зокрема, JPEG-2000 набув офіційного статусу міжнародного стандарту [21,30-34].

4.1 Набір стандартів JPEG 2000

Перелік документів, що становлять повний набір стандартів для JPEG 2000 наведено в табл. 4.1 [30 – 32].

Таблиця 4.1 – Повний набір стандартів JPEG 2000

Частина	Назва	Опис	Номер
Частина 1	Основна система кодування	Визначає синтаксис потоку коду. Різні етапи декодування зображень JPEG 2000. Пояснює базовий формат файлу JP2, метадані та права інтелектуальної власності.	ISO/IEC 15444-1
Частина 2	Розширення	Визначає розширення потоку коду формату файлу і дозволяє демонструвати зразки HDR, специфікацію колірного простору, обрізку, геометричні перетворення; різноманітна анімація, метадані та множинний кодовий потік.	ISO/IEC 15444-2
Частина 3	Motion JPEG 2000 (MJ2 або MJP2)	Введення формату файлу для послідовностей руху, що кодує зображення у незалежному кодовому потоці.	ISO/IEC 15444-3
Частина 4	Відповідність	Вказує методи тестування для кодування та декодування й перевіряє файли як для потоків чистого коду, так і для файлів JP2.	ISO/IEC 15444-4
Частина 5	Еталонне програмне забезпечення	Містить два пакети вихідного коду (Java, C), які реалізують систему кодування Core та доступні за ліцензіями з відкритим вихідним кодом.	ISO/IEC 15444-5
Частина 6	Формат складеного файлу зображення	Визначає формат файлу JPM і дозволяє створювати зображення багатосторінкових документів для програм, подібних до факсу. Підтримує використання JBIG2 та JPEG.	ISO/IEC 15444-6
Частина 8	JPEG 2000 Secured (JPSEC)	Забезпечує безпеку транзакцій, контенту та технологій і дозволяє використовувати захищені бітові потоки JPEG 2000.	ISO/IEC 15444-8
Частина 9	JPIP	Визначає інструменти у мережному середовищі для доступу до метаданих та зображень, а також встановлює інтерактивні й ефективні протоколи	ISO/IEC 15444-9
Частина 10	JP3D	Об'ємне розширення частини 1 та вводить вимірювання Z. Розширює концепцію плиток, кодових блоків, меж та тривимірних областей інтересу.	ISO/IEC 15444-10
Частина 11	JPWL	Розглядає добре організовану передачу за помилкою бездротової мережі. Це розширення сумісне з декодерами	ISO/IEC 15444-11.
Частина 13	Кодер початкового рівня	Визначає реалізацію кодера початкового рівня базової системи кодування.	ISO/IEC 15444-13
Частина 14	JPXML	Подання в XML і пояснює сегменти маркерів та методи доступу до внутрішніх даних зображень.	ISO/IEC 15444-14
Частина 15	HTJ2K (у розробці)	Визначає альтернативний алгоритм блокового кодування. Алгоритм пропонує десятикратно збільшену пропускну здат-ність та кодування/декодування без втрат.	

4.2 Огляд JPEG 2000

Відправною точкою стандарту JPEG 2000 стала пропозиція М. Болієка 1996 року. Розроблений Болієком алгоритм мав стати основою нового стандарту ущільнення зображень без втрат JPEG-LS, але було відкинуто на користь перспективнішого алгоритму LOCO-I. Алгоритм Болієка, проте, мав низку дуже привабливих можливостей, що спричинило створення нового стандарту JPEG 2000.

Оголошення про початок розробки нового стандарту датується березнем 1997 року. Традиційно було влаштовано конкурс алгоритмів ущільнення зображень, на якому проводилося чисельне та візуальне порівняння результатів роботи різних програм. Програма-переможець (нею стала розробка університету Арізони, алгоритм WTCQ) була взята за основу першої версії стандарту. У листопаді 1998 року із подачі Д. Таубмана до стандарту було внесено істотну зміну. Таубман запропонував рішення, що дозволило зробити стандарт значно більш гнучким і менш вимогливим до ресурсів обчислювальної системи. Алгоритм Таубмнана (алгоритм EBCOT) став основою фінальної версії стандарту.

У процесі стандартизації було враховано велику кількість різних пропозицій. Оскільки всі вони не могли скласти новий стандарт, було прийнято рішення частину з них внести в його базовий варіант, а частину розглядати як доповнення. На поточний момент документація стандарту подана двома частинами (у майбутньому плануються додаткові розділи). Перша описує основні моменти, які мають бути обов'язково дотримані в будь-якій реалізації стандарту. Друга містить розширення основної частини стандарту, які є обов'язковими. Цей підхід вигідний тим, що, по-перше, враховує велику кількість різних пропозицій і забезпечує гнучкість, а, по-друге, дозволяє отримувати досить невибагливі щодо обчислювальних ресурсів реалізації, сумісні зі стандартом.

Поділ стандарту на основну та додаткову частину найкраще ілюструється на прикладі алгоритмів квантування. Пропозиція університету Арізони мала на увазі використання складного алгоритму квантування, що отримав назву квантування з сітчастою геометрією. Фактично було запропоновано швидку реалізацію векторного квантування. Як відомо, векторне квантування має більшу ефективність порівняно зі скалярним квантуванням, проте, водночас, воно є значно менш продуктивним. У запропонованому алгоритмі WTCQ векторне квантування реалізовано на базі кінцевого набору скалярних квантувальників, вибір яких (для виконання квантування) здійснюється відповідно до можливих напрямків обходу заданого графіка (мережі). Напрямок обходу одночасно відповідає молодшим розрядам квантованих значень, одержаних з використанням попередніх квантувальників (вибраних на попередньому етапі). Шляхом перебору можливих шляхів

усередині графа можна знайти більш-менш ефективні способи квантування послідовності величин довільної довжини. Незважаючи на те, що подібний алгоритм вельми простий, його складність все ж не йде в жодне порівняння зі складністю алгоритму звичайного рівномірного квантування. Як наслідок, останній (використовується рівномірний квантувач з мертвою зоною поблизу нуля) є обов'язковою частиною стандарту, а перший (квантування з сітчастою геометрією) є розширенням стандарту і є лише його опцією.

Ущільнення за стандартом JPEG-2000 основане на вже класичному алгоритмі пірамідального вейвлет-перетворення. Обробка вейвлет-коефіцієнтів здійснюється методом контекстно-залежного біт-орієнтованого арифметичного кодування.

Спочатку зображення піддається чергуванням послідовностей вертикальних і горизонтальних одновимірних вейвлет перетворень. Першими перетворюються всі рядки, а потім усі стовпці. На наступному етапі ліва верхня чверть матриці, що вийшла внаслідок попереднього перетворення, знову перетворюється (спочатку всі рядки, потім усі стовпці). І так далі. Кількість етапів відповідає кількості рівнів вейв-років-декомпозиції. Внаслідок перетворення ми отримуємо безліч прямокутних діапазонів вейвлет-коефіцієнтів, які прийнято називати частотними діапазонами, оскільки вони містять інформацію про те, як поводиться вихідний двовимірний сигнал (зображення) за різної роздільної здатності (тобто набір коефіцієнтів для різної частоти).

Для перетворення можуть використовуватись різні вейвлет-фільтри. Обов'язкова частина стандарту потребує використання лише двох фільтрів: оборотний «5/3» – для ущільнення без втрат та незворотний «9/3» – для ущільнення з втратами (обидва фільтри є класичними вейвлет-фільтрами). Розширення припускає будь-які інші фільтри. Зрозуміло, що для реалізації перетворення використовується зручна з практичного погляду ліфтинг-схема.

Після перетворення здійснюється квантування коефіцієнтів. Саме на етапі квантування виникають основні інформаційні втрати, і саме за рахунок квантування можливе суттєве зменшення обсягу подання зображення. Звичайно, в квантуванні немає необхідності, якщо проводиться ущільнення без втрат. Як уже було сказано, квантування може бути або рівномірним скалярним, або будь-яким іншим (наприклад, векторним). У разі використання рівномірного скалярного квантування квант-параметр може змінюватись залежно від квантованого діапазону.

Етап арифметичного кодування є завершальним етапом кодування. Діапазони коефіцієнтів розбиваються на прямокутні кодові блоки (як правило, 32×32 або 64×64). Кожен із блоків кодується незалежно. Це означає, що стан арифметичного кодера скидається перед кодуванням чергового кодового блока. У процесі кодування коефіцієнти в блоці віртуально видаються у вигляді бітових площин.

Одна з таких площин – знаки коефіцієнтів; решта площин відповідає різним розрядам величин коефіцієнтів (положення біта в площині відповідає положенню коефіцієнта в блоці). Кодування коефіцієнтів зводиться до кодування бітів, що становлять ці коефіцієнти. Таким чином, арифметичне кодування є біторієнтованим.

Арифметичне кодування ґрунтується на контекстно-залежній моделі. Контекст формується як функція від значень бітів, які оточують біт, що кодується. Кодування здійснюється за площинами: спочатку кодується площина, що відповідає старшому розряду коефіцієнтів, потім наступна за спаданням і т. д. Під час кодування кожному коефіцієнту в блоці, що кодується, ставиться у відповідність параметр «значимість». Коефіцієнт називається значущим, якщо в уже закодованих на цей момент бітових плоскостях присутній хоча б один ненульовий розряд цього коефіцієнта. Кожна бітова площина кодується у три проходи. Під час першого кодового проходу поширюється інформація про значущість коефіцієнтів. Для кожного біта площини, якщо відповідний коефіцієнт ще є значимим, і якщо хоч один сусідній коефіцієнт вже є значущим, здійснюється кодування факту значущості для поточного коефіцієнта, тобто фактично здійснюється кодування значення цього біта поточної площини, що кодується. Якщо біт, що кодується, виявився ненульовим, відразу після його обробки кодується відповідний біт бітової площини знаків коефіцієнтів (кодування знака). Під час другого кодового проходу кодуються всі біти, що відповідають значимим на цей момент коефіцієнтам і не оброблялися під час першого проходу. На відміну від попереднього кодового проходу, коли рішення про кодування приймалося на основі інформації про значущість сусідніх коефіцієнтів, під час проходу біти кодуються в обов'язковому порядку. Мета третього кодового проходу – обробити ті біти, які не були оброблені під час першого та другого проходів. Під час третього проходу арифметичне кодування застосовується разом із груповим кодуванням. Істотною деталлю, передбаченою стандартом, є можливість пропуску кодових проходів, що є ще одним джерелом підвищення ефективності за рахунок інформаційних втрат (першим найбільш явним джерелом є квантування). Ця можливість активно використовується для здійснення контролю за швидкістю генерації коду.

Подання інформації, отримане внаслідок вейвлет-перетворення, дуже зручне тим, що воно забезпечує можливість отримання приблизних копій зображення без здійснення повного обернене перетворення. Обернене перетворення здійснюється у порядку, зворотному порядку прямого перетворення. Виробляючи обмежену кількість обернених декомпозицій (об'єднання частотних діапазонів), вважаючи, що всі частоти, які не залучені до перетворення, містять виключно нульові елементи, ми легко можемо отримати або копію зображення в зменшеному масштабі, або вихідне зображення, але більш низької якості порівняно із зображенням,

отриманим як результат повного оберненого перетворення. Враховуючи той факт, що блоки вейвлет-коефіцієнтів кодуються незалежно один від одного, ми отримуємо можливість часткового декодування не лише на рівні перетворення, а й на рівні інтерпретації коду. Для отримання приблизної копії зображення достатньо декодувати всього лише частину інформації, а потім здійснити часткове обернене перетворення.

Іншою важливою перевагою нового стандарту є можливість доступу до окремих елементів зображення без повного декодування його подання. Забезпечується така можливість, по-перше, розбиттям вихідного зображення на області, що не перетинаються (тайли), які кодуються як окремі зображення, а по-друге, поданням коду окремого тайла у вигляді частин (шарів), кожна з яких є сумарним кодом коефіцієнтів, що відповідають деякій його (тайлу) області (зазначимо, що шари також діляться на так звані пакети, що містять код блоків коефіцієнтів на різних рівнях декомпозиції). Для того, щоб декодувати будь-яку область зображення, достатньо визначити, яким тайлам вона належить і які шари, що відносяться до цих тайлів, містять код блоків коефіцієнтів, необхідних для відновлення необхідної області.

Безумовно, «зручне» зображення не може бути вигідним з погляду ефективності ущільнення. Справді, зі зменшенням розміру структурних елементів (тайлів, областей тайлів, що утворюють шари та ін.), ефективність ущільнення дещо знижується. Стандарт у цьому випадку залишає нам вибір: з однієї сторони, ми маємо можливість отримувати інформаційні подання, що дозволяють досить швидко отримувати та редагувати частини зображення, з іншого боку, стандарт не перешкоджає створенню інформаційних подань, ефективних за обсягом.

Як неважко помітити, все, що було сказано вище, насправді стосується не зовсім ущільнення зображень. Йшлося лише про ущільнення матриць. Реальні зображення часом є складнішими об’єктами. Як правило, зображення включає відразу декілька компонентів. Найчастіше воно складається з трьох кольорових компонентів: червоного, зеленого та синього. Оскільки кожен компонент окремо є матрицею, для того, щоб закодувати зображення повністю, нам необхідно закодувати не одну, а три матриці. Такий підхід, як показує практика, прийнятний, але не є вдалим. Більшої ефективності ущільнення можна досягти у випадку, коли компоненти, що кодуються, представлені в яскраво-колірній формі.

Для перетворення зображення зі стандартного кольорового представлення RGB в яскраво-колірне представлення YCrCb стандартом передбачено дві процедури: оборотна та необоротна. Необоротна процедура точно повторює класичне перетворення RGB-> YCrCb, яке використовувалося, наприклад, у старому стандарті JPEG. Оборотна процедура є досить грубе наближення до класичної незворотної процедури. Як випливає з назви, це перетворення не веде до втрати

колірної інформації, і може застосовуватися в тих випадках, коли використовується режим ущільнення без втрат.

Для забезпечення завадостійкості інформаційного подання та зручності доступу до інформації у стандарті JPEG 2000 передбачено систему маркерів та маркерних сегментів. Маркери відіграють роль розмежувачів усередині інформаційного потоку; маркерні сегменти містять параметри фрагментів інформації обмежених маркерами. Дані, що починаються з маркера, як правило, можуть бути коректно проінтерпретовані без будь-якої додаткової інформації (це, природно, не означає можливість відновлення цілого за фрагментами), що забезпечує можливість часткового відновлення зображення, подання якого було пошкоджено. Введення елементів завадостійкості дає зелене світло використанню стандарту у різноманітних телекомунікаційних додатках.

На закінчення огляду стандарту потрібно сказати кілька слів щодо його реальної ефективності. Досягнення високої якості ущільнення, безумовно, було одним із головних завдань під час його створення, і тут розробники досягли явного прогресу. Стандарт JPEG 2000 перевищує ефективність JPEG приблизно в 2 рази у випадку ущільнення з втратами і на 5 – 20% у разі ущільнення без втрат. Звичайно, ефективність у разі ущільнення без втрат у цьому випадку виявляється не такою високою, як, скажімо, стандарту JPEG-LS, проте вона цілком прийнятна. Що ж стосується ефективності ущільнення з втратами, тут стандарт дозволяє отримувати результати, близькі до найкращих, на сьогодні, результатів для таких методів.

Судити про швидкість роботи практичних реалізацій стандарту поки ще рано – кількість таких реалізацій на сьогодні дуже невелика. Проте можна сказати, що, як і очікувалося, новий стандарт дещо повільніший за свого попередника.

JPEG2000, безперечно, має рано чи пізно стати повноцінним стандартом, чому є всі передумови. Високий ступінь ущільнення, можливість швидкого доступу до окремих частин інформації, чудова структурованість, невибагливість до ресурсів та багато іншого – все це сприятиме поширенню нового стандарту і, можливо, стане сильним поштовхом для подальшого розвитку технологій обробки відеоінформації [35, 36].

4.3 Рекурсивний (хвильовий) алгоритм ущільнення зображень

4.3.1 Пірамідальне перетворення

Англійська назва рекурсивного ущільнення – wavelet. Цей алгоритм орієнтований на ущільнення кольорових і чорно-білих зображень з плавними переходами, ідеальний для картинок типу рентгенівських фотографій. Коефіцієнт ущільнення варіюється в межах 5 – 100. За великих коефіцієнтів ущільнення на різких границях, особливо діагональних, можливі спотворення [4].

Основна ідея алгоритму полягає в тому, що зберігаються різниці між середніми значеннями сусідніх блоків зображення, які звичайно набувають значень, близьких до нуля.

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

Ущільнення 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 ущільнення необхідно зниження точності подання компонент зображення, отриманих внаслідок виконання S-перетворення, але так, щоб спотворення не були візуально помітні.

Структуру кодування наведено на рис. 4.1. Вхідні відеодані 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 мають бути непомітними, тобто візуальна якість відновленого зображення не може відрізнятися від вхідного.

Власне ущільнення може бути виконано на основі одного з методів статистичного кодування (наприклад, кодування Гаффмана, LZW-кодування, арифметичного кодування).

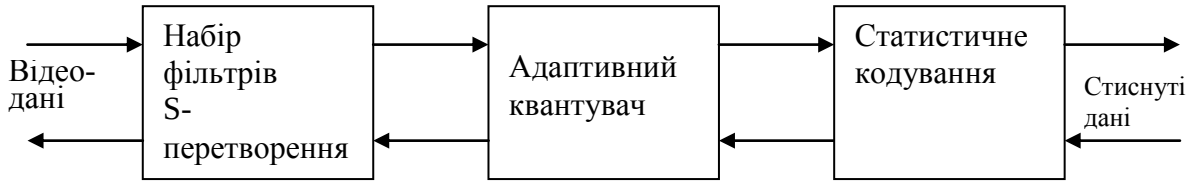


Рисунок 4.1 – Схема хвильового кодування

Основну схему пірамідального S-перетворення наведено на рис. 4.2, а оберненого перетворення – на рис. 4.3.

Відліки низько- (фільтр F1) і високочастотної (фільтр F2) складової одержують відповідно до таких виразів:

$$\begin{aligned} L(n) &= [C(2*n)+C(2*n+1)]/2 \\ H(n) &= [C(2*n)-C(2*n+1)]/2 \end{aligned} \qquad (4.1)$$

У цих виразах n= 0, 1... (N/2 – 1), C [2*n], C [2*n + 1] – відліки вхідного цифрового сигналу, а L[n], H[n] – низькочастотні і високочастотні коефіцієнти S-перетворення (рис. 4.2), L[n] – відліки на виході низькочастотного фільтра, H[n] – відліки на виході високочастотного

фільтра. Ділення на два може бути виконано зсувом вправо на один біт, що дає результат з округленням до меншого.

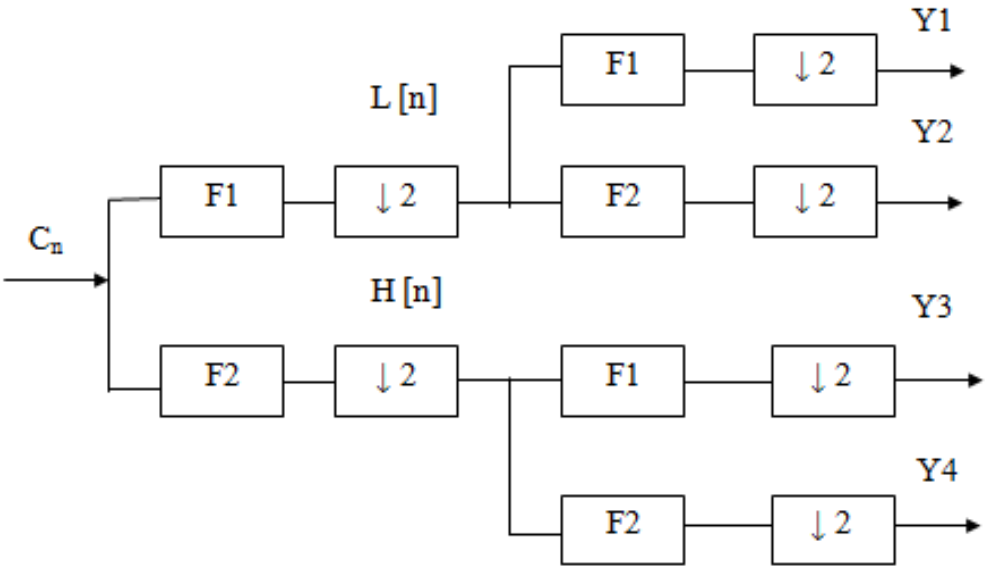
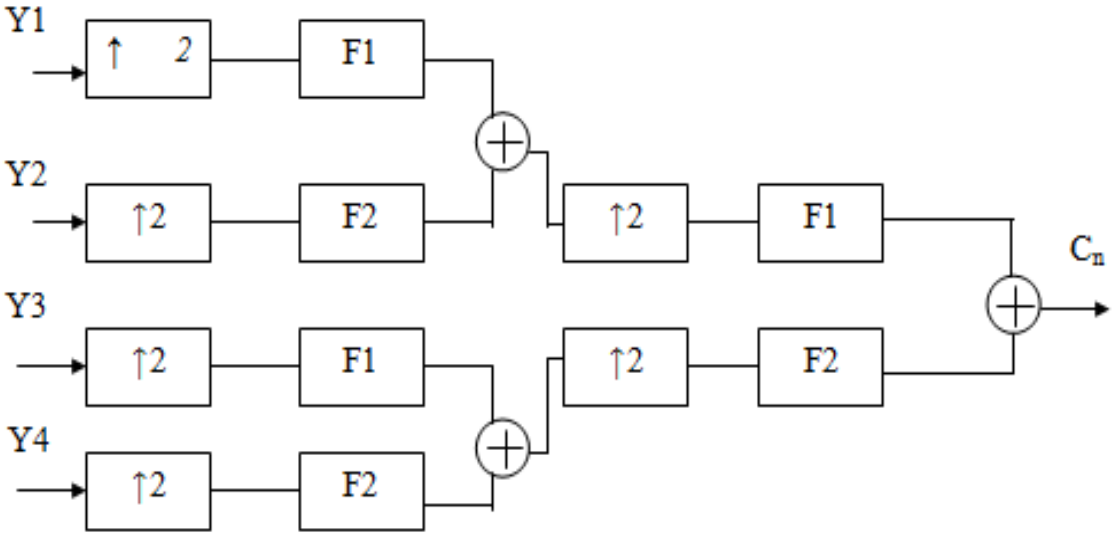


Рисунок 4.2 – Пряме S-перетворення



65

Рисунок 4.3 – Обернене S-перетворення

До кожного вихідного компонента фільтрів знову застосовуються дані формули, утворюючи в такий спосіб піраміду. Ясно, що чим більша кількість компонентів, тим більший очікуваний коефіцієнт ущільнення.

Перевагою даного представлення вихідного зображення є те, що дисперсія H[n] менша ніж для C [n].

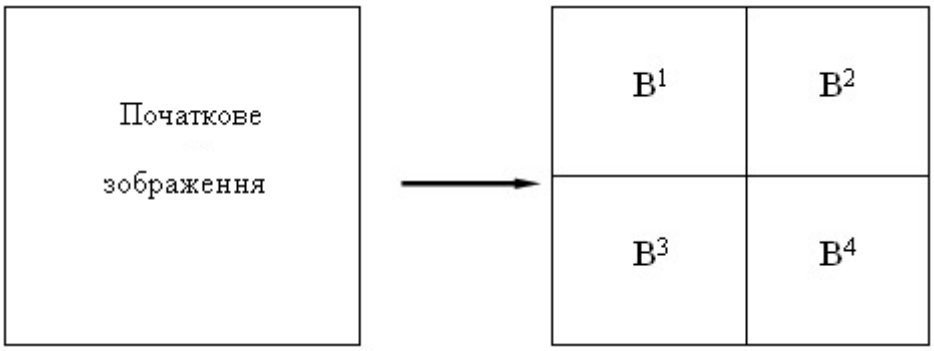
Інверсне S-перетворення обчислюється в такий спосіб:

$$\begin{aligned} C(2*n) &= L(n) + H(n), \\ C(2*n+1) &= L(n) - H(n) \end{aligned} \qquad (4.2)$$

Алгоритм для двовимірних даних реалізується аналогічно. Для квадрата з розмірами 2×2 і значеннями елементів $a_{2i,2j}$, $a_{2i+1,\,2j}$, $a_{2i,2j+1}$, $a_{2i+1,2j+1}$ отримаємо

$$\begin{aligned} b^1_{i,j} &= (a_{2i,2j} + a_{2i+1,2j} + a_{2i,2j+1} + a_{2i+1,2j+1})/4 \\ b^2_{i,j} &= (a_{2i,2j} + a_{2i+1,2j} - a_{2i,2j+1} - a_{2i+1,2j+1})/4 \\ b^3_{i,j} &= (a_{2i,2j} - a_{2i+1,2j} + a_{2i,2j+1} - a_{2i+1,2j+1})/4 \\ b^4_{i,j} &= (a_{2i,2j} - a_{2i+1,2j} - a_{2i,2j+1} + a_{2i+1,2j+1})/4 \end{aligned} \qquad (4.3)$$

Використовуючи ці формули, наприклад, для зображення з розмірами 512×512 пікселів після першого перетворення одержимо 4 матриці розміром 256×256 елементів:



У першій, як легко здогадатися, буде зберігатися зменшена копія зображення.

В другій – усереднені різниці пар значень пікселів по горизонталі.

У третій – усереднені різниці пар значень пікселів по вертикалі.

У четвертій – усереднені різниці значень пікселів по діагоналі.

За аналогією з двовимірним випадком, ми можемо повторити наше перетворення й одержати замість першої матриці 4 матриці розміром 128×128.

Повторивши наше перетворення втретє, ми одержимо в підсумку: 4 матриці 64×64, 3 матриці 128×128 і 3 матриці 256×256.

На практиці, під час записування у файл, значення B^4 звичайно не враховують, відразу одержуючи виграш приблизно на третину розміру файлу.

Можливі варіанти розкладу зображення на компоненти наведено на рис. 4.4 – 4.11.

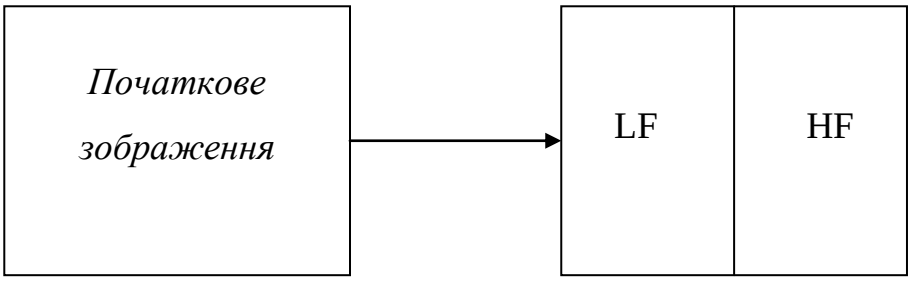


Рисунок 4.4 – Горизонтальне двокомпонентне кодування. LF – низькочастотні коефіцієнти S-перетворення, HF – високочастотні коефіцієнти

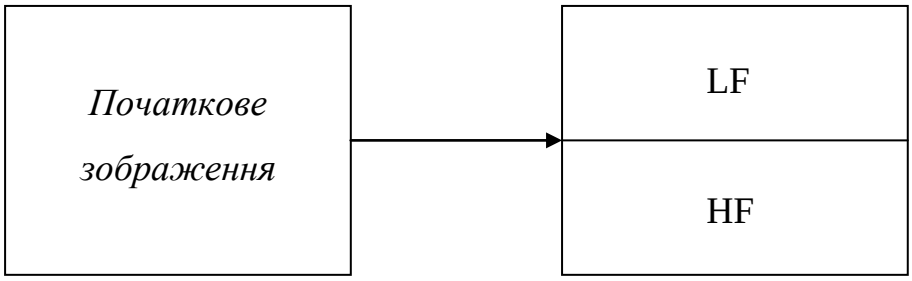


Рисунок 4.5 – Вертикальне двокомпонентне кодування. LF – низькочастотні коефіцієнти S-перетворення, HF – високочастотні коефіцієнти

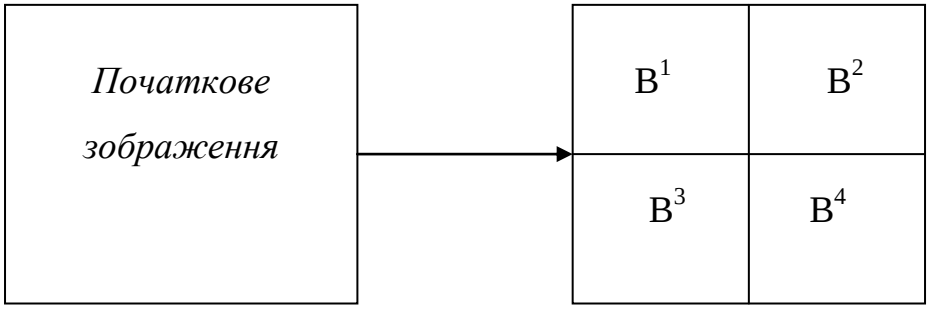


Рисунок 4.6 – Двовимірне чотирикомпонентне кодування

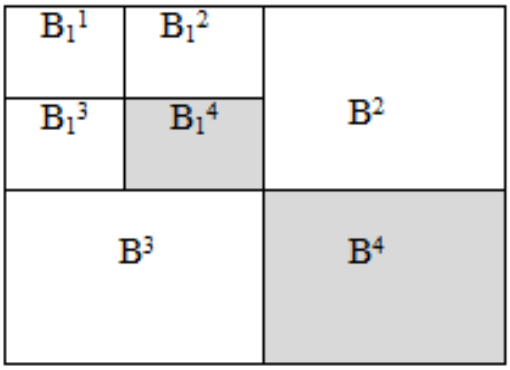


Рисунок 4.7 – Двовимірне семикомпонентне кодування

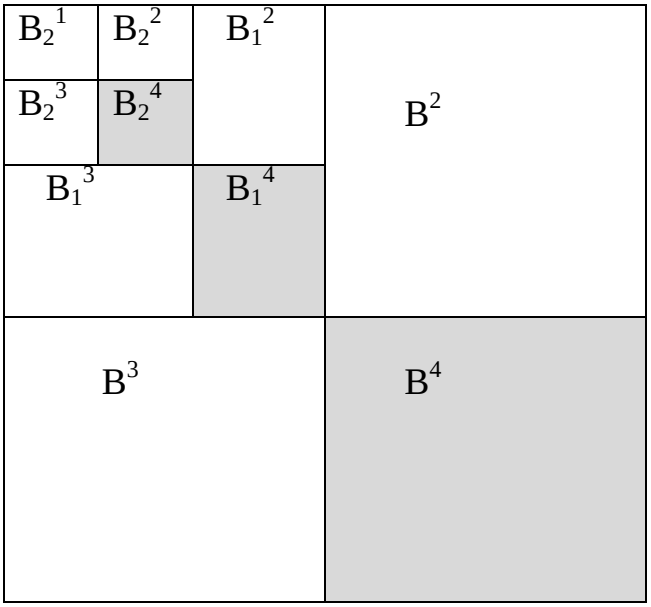


Рисунок 4.8 – Двовимірне десятикомпонентне кодування

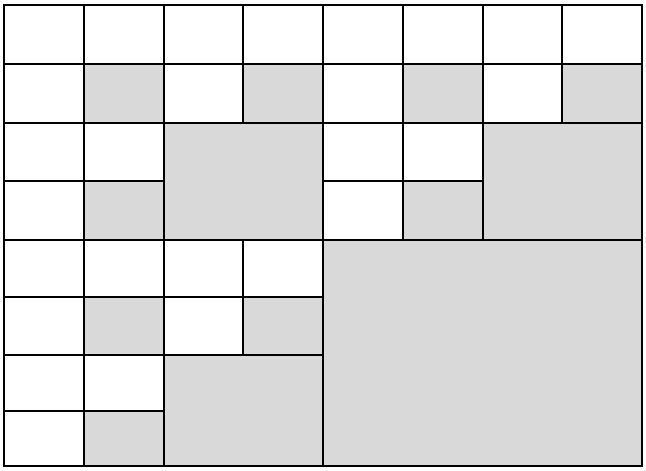
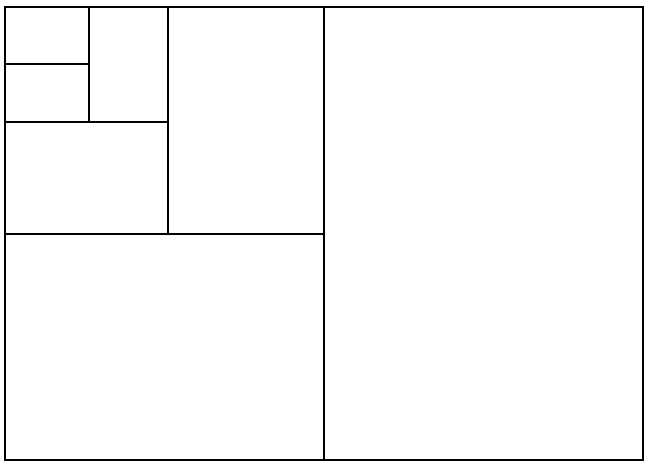
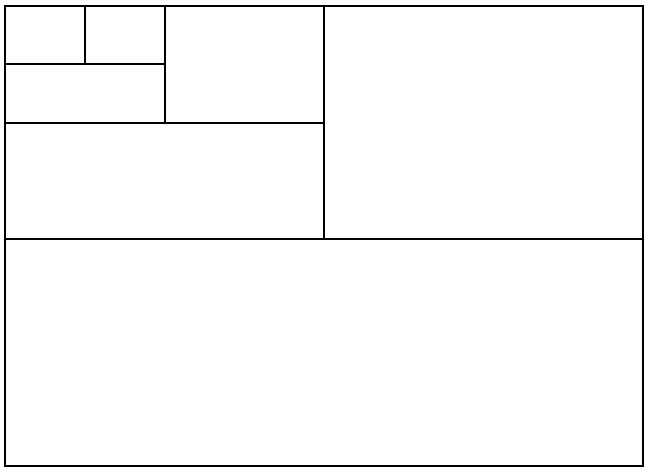


Рисунок 4.9 – Двовимірне 40-компонентне кодування



а



б

Рисунок 4.10 – Альтернативні варіанти виконання S-перетворення

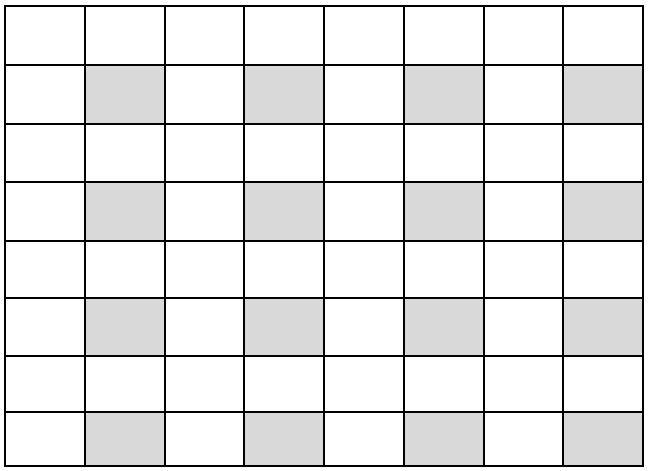


Рисунок 4.11 – Повний 64-компонентний розклад

До переваг цього алгоритму можна віднести те, що він дуже легко дозволяє реалізувати можливість поступового «прояву» зображення під час передачі зображення по мережі. Крім того, оскільки на початку зображення ми фактично зберігаємо його зменшену копію, спрощується показ «огрубленого» зображення по заголовку.

Двовимірне 2D-перетворення може бути також виконане за допомогою застосування формул (4.1) для одновимірного перетворення послідовно до рядків і стовпців зображення. Це зменшує обчислювальні затрати, необхідні для виконання цього перетворення. Сигнальний граф швидкого двовимірного S-перетворення для фрагмента 2×2, наведено на рис. 4.12.

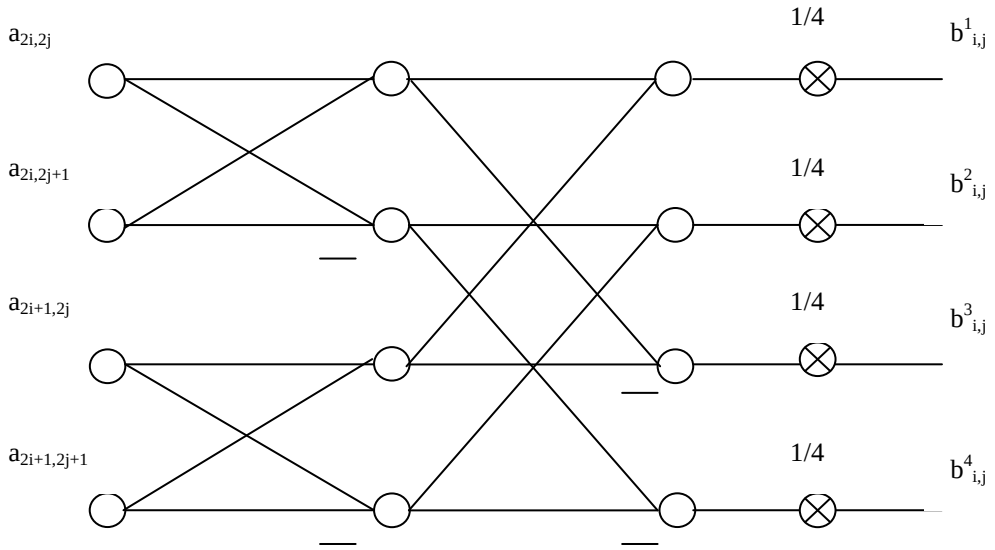


Рисунок 4.12 – Сигнальний граф швидкого двовимірного S-перетворення для фрагмента 2×2

Зображення, які ілюструють етапи Wavelet-кодування приведені на рис. 4.13-4.16.

Рисунок 4.13 – Компоненти зображення

Рисунок 4.14 – Відновлене зображення майже без втрат

Рисунок 4.15 – Розкладення без компоненти B^4

Рисунок 4.16 – Відновлене зображення без компоненти B^4

На відміну від JPEG і фрактального алгоритму, S-перетворення не оперує з блоками, наприклад, 8×8 пікселів, а оперує блоками 2×2, 4×4, 8×8 і т. д. Однак за рахунок того, що коефіцієнти для цих блоків ми зберігаємо незалежно, ми можемо досить легко уникнути дроблення зображення на «мозаїчні» квадрати.

Розглянемо конкретний приклад: нехай ми перетворюємо рядок із 8 значень яскравості пікселів:

(a_i): (100, 110, 200, 200, 220, 217, 212, 210).

Ми отримаємо такі послідовності b_i^1 , b_i^2:

(105, 200, 218.5, 211) та (-5, 0, 1.5, 1).

Зауважимо, що значення b_i^2 досить близькі до 0. Повторимо операцію, розглядаючи b_i^1 як a_i. Ця дія виконується ніби рекурсивно, звідки і назва алгоритму. Ми отримаємо:

(105, 200, 218.5, 211): (152.5, 214.75) (-47.5, 3.75).

Отримані коефіцієнти, округливши до цілих і стиснувши, наприклад, з допомогою алгоритму Гаффмана з фіксованими таблицями, можемо помістити файл.

Зауважимо, що ми застосовували наше перетворення до ланцюжка лише двічі. Реально ми можемо дозволити собі застосування wavelet-перетворення 4-6 разів. Більше того, додаткове ущільнення можна

отримати, використовуючи таблиці алгоритму Гаффмана з нерівномірним кроком (тобто нам доведеться зберігати код Гаффмана для найближчого в таблиці значення).

Характеристики хвильового алгоритму:
Ступінь: 2-200 (Задається користувачем).
Клас зображень: Як у фрактального та JPEG
Симетричність: ~1.5
Характерні риси: Крім того, при високому коефіцієнті ущільнення зображення розпадається на окремі блоки.

4.4 Алгоритм JPEG 2000

4.4.1 Базова схема JPEG 2000

Алгоритм JPEG-2000 розроблений тією самою групою експертів у галузі фотографії, що й JPEG. Формування JPEG як міжнародного стандарту було закінчено у 1992 році. У 1997 стало ясно, що потрібен новий, гнучкіший і потужніший стандарт, який і був доопрацьований до зими 2000 року. Основні відмінності алгоритму в JPEG 2000 від алгоритму в JPEG полягають у такому:

1. **Найкраща якість зображення за сильного ступеня ущільнення**. Або, що те саме, великий ступінь ущільнення за збереження високої якості. Фактично це означає помітне зменшення розмірів графіки «Web-якості», яка використовується більшістю сайтів.
2. **Підтримка кодування окремих областей із найкращою якістю**. Відомо, що окремі області зображення є критичними для сприйняття людиною (наприклад, очі на фотографії), у той час як якістю інших можна пожертвувати (наприклад, задній план). У випадку «ручної» оптимізації збільшення ступеня ущільнення проводиться доти, доки не буде втрачено якість у якійсь важливій частині зображення. Сьогодні з'являється можливість задати якість у критичних областях, ущільнивши інші області сильніше, тобто ми отримуємо ще більший остаточний ступінь ущільнення за суб'єктивно однакової якості зображення.
3. **Основний алгоритм ущільнення замінений на Wavelet.** Крім зазначеного підвищення ступеня ущільнення, це дозволило позбавитися від 8-пікселної блочності, що виникає у разі підвищення ступеня ущільнення.
4. **Для підвищення ступеня ущільнення в алгоритмі використовується арифметичне кодування**. Спочатку в стандарті JPEG також було закладено арифметичне стиснення, проте пізніше воно було замінено менш ефективним стисненням за Гаффманом, оскільки арифметичне кодування було захищено патентами. Зараз термін дії основного патенту минув, і з'явилася можливість покращити алгоритм.

5. **Підтримка ущільнення без втрат**. Крім звичного ущільнення з втратами, новий JPEG тепер буде підтримувати і ущільнення без втрат. Таким чином, стає можливим використання JPEG для ущільнення медичних зображень, у поліграфії, під час збереження тексту під розпізнавання OCR системами і т. д.
6. **Підтримка ущільнення однобітних (2-колірних) зображень**. Для збереження однобітних зображень (малюнки тушшю, відсканований текст і т. п.) раніше повсюдно рекомендувався формат GIF, оскільки ущільнення з використанням ДКП дуже неефективно до зображень з різкими переходами кольорів. В JPEG під час ущільнення 1-бітна картинка зводилась до 8-бітної, тобто збільшувалася у 8 разів, після чого робилася спроба ущільнювати, нерідко менш ніж у 8 разів. Зараз можна рекомендувати JPEG 2000 як універсальний алгоритм.
7. **На рівні формату підтримується прозорість**. Плавно накладати фон в процесі створення WWW сторінок тепер можна буде не тільки в GIF, а й у JPEG 2000. Крім того, підтримується не лише 1 біт прозорості (піксел прозорий/непрозорий), а окремий канал, що дозволить задавати плавний перехід від непрозорого зображення до прозорого фону.

Крім того, на рівні формату підтримуються включення до зображення інформації про копірайт, підтримка стійкості до бітових помилок під час передавання та широкомовлення, можна запитувати для декомпресії або обробки зовнішні засоби (plug-ins), можна вносити в зображення його опис, інформацію для пошуку і т. д.

Ідея алгоритму

Базова схема JPEG-2000 (рис. 4.17) дуже схожа на базову схему JPEG. Відмінності полягають у такому [35 – 36]:

1. Замість дискретного косинусного перетворення (DCT) використовується дискретне вейвлет-перетворення (DWT).
2. Замість кодування за Гаффманом використовується арифметичне кодування.
3. В алгоритм спочатку закладено управління якістю областей зображення.
4. Не використовується явно дискретизація компонентів U і V після перетворення колірних просторів, оскільки при DWT можна досягти того ж результату, але акуратніше.

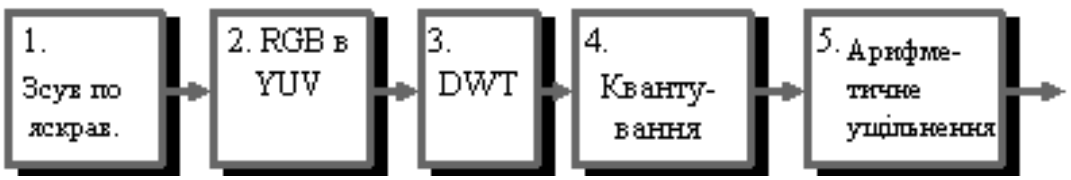


Рисунок 4.17 – Базова схема JPEG 2000

4.4.2 Алгоритм виконання кроків JPEG-2000

Крок 1.

У JPEG 2000 передбачено зсув яскравості (DClevelshift) кожної компоненти (RGB) зображення перед перетворенням на YUV. Це робиться для вирівнювання динамічного діапазону (наближення до 0 гістограми частот), що приводить до збільшення ступеня ущільнення. Формулу перетворення можна записати як:

$$I'(x,y) = I(x,y) - 2^{ST-1} \tag{4.4}$$

Значення ступеня ST для кожної компоненти R, G і B своє (визначається в процесі ущільнення компресором). Під час відновлення зображення виконується обернене перетворення:

$$I(x,y) = I'(x,y) + 2^{ST-1} \tag{4.5}$$

Крок 2.

Перетворюємо зображення з колірного простору RGB з компонентами, що відповідають за червону (Red), зелену (Green) та синю (Blue) складові кольору точки, в колірний простір YUV. Цей крок аналогічний JPEG (Див. матриці перетворення в описі JPEG), за винятком, що крім перетворення з втратами передбачено також і перетворення без втрат. Його матриця виглядає так:

$$\begin{vmatrix} Y \\ U \\ V \end{vmatrix} = \begin{pmatrix} \left\lfloor \frac{R + 2G + B}{4} \right\rfloor \\ R - G \\ B - G \end{pmatrix} \tag{4.6}$$

Зворотне перетворення визначається так:

$$\begin{vmatrix} R \\ G \\ B \end{vmatrix} = \begin{pmatrix} U + G \\ Y - \left\lfloor \frac{U + V}{4} \right\rfloor \\ V + G \end{pmatrix} \tag{4.7}$$

Крок 3.

Дискретне wavelet перетворення (DWT) може бути двох видів - для випадку ущільнення з втратами та для ущільнення без втрат. Його коефіцієнти задаються таблицями, наведеними нижче. Для ущільнення з втратами коефіцієнти представлені в табл. 4.2 і табл. 4.3. Для ущільнення без втрат коефіцієнти представлені в табл. 4.4.

Саме перетворення в одновимірному випадку є скалярним добутком коефіцієнтів фільтра на рядок перетворюваних значень (у нашому випадку – на рядок зображення).

Таблиця 4.2 – Коефіцієнти DWT під час упаковки при ущільненні з втратами

i	Низькочастотні коефіцієнти $h_L(i)$	Високочастотні коефіцієнти $h_H(i)$
0	1.115087052456994	0.6029490182363579
±1	0.5912717631142470	-0.2668641184428723
±2	-0.05754352622849957	-0.07822326652898785
±3	-0.09127176311424948	0.01686411844287495
±4	0	0.02674875741080976
Інші i	0	0

Таблиця 4.3 – Коефіцієнти DWT під час розпаковки при ущільненні з втратами

i	Низькочастотні коефіцієнти $g_L(i)$	Високочастотні коефіцієнти $g_H(i)$
0	0.6029490182363579	1.115087052456994
±1	-0.2668641184428723	0.5912717631142470
±2	-0.07822326652898785	-0.05754352622849957
±3	0.01686411844287495	-0.09127176311424948
±4	0.02674875741080976	0
Інші i	0	0

Таблиця 4.4 – Коефіцієнти DWT при ущільненні без втрат

	При упаковці		**При розпаковуванні**	
i	Низькочастотні коефіцієнти $h_L(i)$	Високочастотні коефіцієнти $h_H(i)$	Низькочастотні коефіцієнти $g_L(i)$	Високочастотні коефіцієнти $g_H(i)$
0	6/8	1	1	6/8
±1	2/8	-1/2	½	-2/8
±2	-1/8	0	0	-1/8

У цьому випадку парні вихідні значення формуються за допомогою низькочастотного перетворення, а непарні за допомогою високочастотного:

$$y_{out}(2n) = \sum_{i=-M}^{M} x_{in}(2n-i)\cdot h_L(i)$$

$$y_{out}(2n+1) = \sum_{i=-M}^{M} x_{in}(2n+1-i)\cdot h_H(i) \qquad (4.8)$$

$$n = 0,1,\ldots\frac{N}{2}-1;\ N\text{ - кількість відліків};\ M\text{ - порядок фільтру.}$$

Під час оберненого перетворення навпаки – парні вихідні значення формуються за допомогою високочастотного перетворення, а непарні – за допомогою низькочастотного:

$$x_{out}(2n) = \sum_{i=-M}^{M} y_{out}(2n-i)\cdot g_H(i)$$

$$x_{out}(2n+1) = \sum_{i=-M}^{M} y_{out}(2n+1-i)\cdot g_L(i) \qquad (4.9)$$

$$n = 0,1,\ldots\frac{N}{2}-1;\ N\text{ - кількість відліків};\ M\text{ - порядок фільтру.}$$

Оскільки більшість коефіцієнтів, крім окóлу i=0, дорівнюють 0, то можна переписати наведені формули з меншою кількістю операцій.

Для простоти розглянемо випадок ущільнення без втрат.

$$y_{out}(2n) = \frac{-x_{in}(2n-2)+2\cdot x_{in}(2n-1)+6\cdot x_{in}(2n)+2\cdot x_{in}(2n+1)-x_{in}(2n+2)}{8}$$

$$y_{out}(2n+1) = -\frac{x_{in}(2n)}{2}+x_{in}(2n+1)-\frac{x_{in}(2n+2)}{2} \qquad (4.10)$$

Легко показати, що цей запис можна еквівалентно переписати, зменшивши ще втричі кількість операцій множення та ділення (проте тепер необхідно буде підрахувати спочатку всі непарні y). Додамо також операції округлення до найближчого цілого, що не перевищує задане число a, що позначаються як $\lfloor a \rfloor$:

$$y_{out}(2n+1) = x_{in}(2n+1)-\left\lfloor\frac{x_{in}(2n)+x_{in}(2n+2)}{2}\right\rfloor \qquad (4.11)$$

$$y_{out}(2n) = x_{in}(2n)+\left\lfloor\frac{y_{out}(2n-1)+y_{out}(2n+1)}{4}\right\rfloor$$

$$n = 0,1,\ldots\frac{N}{2}-1$$

Вправа: Самостійно зменшіть кількість операцій без втрат.

Розглянемо на прикладі, як працює це перетворення. Для того, щоб перетворення можна було застосовувати до крайніх пікселів зображення, воно симетрично добудовується в обидві сторони на кілька пікселів, як показано на рис. 4.18. Для ущільнення із втратами нам необхідно добудувати зображення на 4 пікселі, а для ущільнення без втрат на 2 піксела.

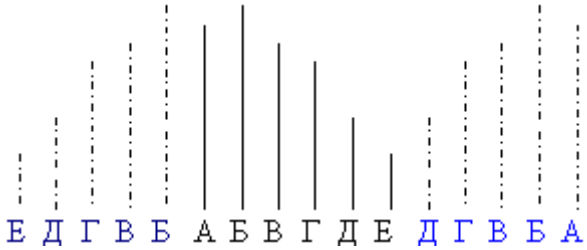


Рисунок 4.18 – Симетричне розширення зображення (яскравості АБ…Е) по рядку вправо та вліво

Нехай потрібно перетворити рядок із 10 пікселів. Розширимо її значення праворуч і ліворуч та застосуємо DWT перетворення:

n	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
x_{in}	3	2	1	2	3	7	10	15	12	9	10	5	10	9
y_{out}		0	1	0	3	1	11	4	13	-2	8	-5		

Утворений рядок 1, 0, 3, 1, 11, 4, 13, -2, 8, -5 і є ланцюжком, що однозначно задає вихідні дані. Здійснивши аналогічні перетворення з коефіцієнтами для розпакування, наведеними вище в таблиці, отримаємо необхідні формули:

$$x_{out}(2n) = y_{out}(2n) - \left\lfloor \frac{y_{out}(2n-1) + y_{out}(2n+1)}{4} \right\rfloor \qquad (4.12)$$

$$x_{out}(2n+1) = y_{out}(2n+1) + \left\lfloor \frac{x_{out}(2n) + x_{out}(2n+2)}{2} \right\rfloor$$

Вправа. Доведіть, що у всіх випадках округлення ми отримуватимемо однакові вхідні та вихідні ланцюжки.

Легко перевірити (використовуючи перетворення упаковки), що значення на кінцях рядків у y_{out} також симетричні відносно n=0 і 9. Скориставшись цією властивістю, розширимо наш рядок праворуч і ліворуч та застосуємо обернене перетворення:

n	-2	-1	0	1	2	3	4	5	6	7	8	9	10	11
y_{out}		0	1	0	3	1	11	4	13	-2	8	-5	8	-2
x_{out}			1	2	3	7	10	15	12	9	10	5	10	

Вихідний ланцюжок $x_{in}=x_{out}$.

Далі до рядка застосовується надрядкове перетворення, суть якого полягає в тому, що всі парні коефіцієнти переписуються на початок рядка, а всі непарні - на кінець. Внаслідок цього перетворення на початку рядка формується «зменшена копія» всього рядка (низькочастотна складова), а наприкінці рядка - інформація про коливання значень проміжних пікселів (високочастотна складова).

y_{out}	1	0	3	1	11	4	13	-2	8	-5
y_{out}'	1	3	11	13	8	0	1	4	-2	-5

Для зображень, які є двовимірним масивом, це перетворення застосовується спочатку до всіх рядків зображення, а потім до всіх колонок зображення. Як результат – зображення ділиться на 4 квадранти (приклади дивіться в описі рекурсивного ущільнення). У першому квадранті буде сформовано зменшену копію зображення, а в решті трьох – високочастотну інформацію. Після чого перетворення повторно застосовується вже тільки до першого квадранту зображення за тими самими правилами (див. рис. 4.13).

Для коректного збереження результатів під дані 2 та 3 квадрантів виділяється на один біт більше, а під дані 4-го квадранта – на 2 біти більше. Тобто. якщо початкові дані були 8-бітні, то на 2 та 3 квадранти потрібно 9 біт, а на 4-й - 10, незалежно від рівня застосування DWT. Під час записування коефіцієнтів у файл можна використовувати ієрархічну структуру DWT, розміщуючи коефіцієнти перетворень з більшого рівня на початок файлу. Це дозволяє отримати «зображення для перегляду», прочитавши невелику ділянку даних з початку файлу, а не розпаковуючи весь файл, як це доводилося робити у випадку ущільнення зображення в JPEG. Ієрархічність перетворення може також використовуватися для плавного покращення якості зображення у разі передачі його через мережу.

Крок 4.

Так само, як і в алгоритмі JPEG, після DWT застосовується квантування. Коефіцієнти квадрантів діляться на заздалегідь задане число. За збільшення цього числа знижується динамічний діапазон коефіцієнтів, вони стають ближчими до 0, і ми отримуємо більший ступінь ущільнення. Варіюючи ці числа для різних рівнів перетворення, для різних колірних компонентів і для різних квадрантів, ми можемо дуже гнучко керувати ступенем втрат у зображенні. Розраховані в компресорі оптимальні коефіцієнти квантування передаються декомпресору для однозначного розпакування.

Крок 5.

Для ущільнення масивів даних після wavelet перетворення, в JPEG 2000 використовується варіант арифметичного кодування, відомий як MQ-кодер, прообраз якого (QM-кодер) розглядався ще у стандарті JPEG, але реально не використовувався через патентні обмеження. Докладніше про алгоритм арифметичного кодування читайте в [11].

4.5 Області підвищеної якості

Основне завдання, яке ми вирішуємо, – підвищення ступеня ущільнення зображень. Коли практично досягнуто межу ущільнення зображення загалом і різні методи дають дуже невеликий виграш, ми можемо суттєво (в рази) збільшити ступінь ущільнення за рахунок зміни якості різних ділянок зображення.

Проблемою цього підходу є те, що необхідно якимось чином знати розташування найбільш важливих для людини ділянок зображення. Якщо під час ущільнення фотографії людини з великими втратами буде розмито обличчя чи очі, то експертна оцінка якості зображення буде гіршою (рис. 4.19), але для предметів, що знаходяться на задньому плані - це несуттєво.

Рисунок 4.19 - Локальне покращення якості областей зображення

Роботи з автоматичного виділення таких областей активно ведуться. Зокрема, створено алгоритми автоматичного виділення облич на зображеннях. Продовжуються дослідження методів виділення найзначніших (під час аналізування зображення мозком людини) контурів і т. д. Однак очевидно, що універсальний алгоритм найближчим часом створено не буде, оскільки для цього потрібно побудувати повну схему сприйняття зображень мозком людини.

На сьогодні цілком реальне застосування напівавтоматичних систем, в яких якість областей зображення задаватиметься інтерактивно. Цей підхід

зменшує кількість можливих областей застосування модифікованого алгоритму, але дозволяє досягти більшого ступеня ущільнення.

Такий підхід логічно застосовувати, якщо:

1. Для застосунку критичним є ступінь ущільнення, причому настільки, що можливий індивідуальний підхід до кожного зображення.
2. Зображення стискається один раз, а розтискається багато разів.

Як приклади додатків, що задовольняють ці обмеження, можна навести практично всі мультимедійні продукти на CD-ROM. І для CD-ROM енциклопедій, і для ігор важливо записати на диск якнайбільше інформації, а графіка, як правило, займає до 70% всього обсягу диска. У цьому випадку технологія виробництва дисків дозволяє ущільнювати кожне зображення індивідуально, максимально підвищуючи ступінь ущільнення..

Цікавим прикладом є WWW-сервер. Для них теж, як правило, виконуються обидві наведені вище умови. Водночас абсолютно не обов'язково індивідуально підходити до кожного зображення, оскільки за статистикою 10% зображень будуть запитуватись 90% разів. Тобто. для великих довідкових або ігрових серверів з’являється можливість зменшувати час завантаження зображень і рівень завантаженості каналів зв'язку адаптивно.

У JPEG 2000 використовується однобітне зображення-маска, що задає підвищення якості у певних областях зображення. Оскільки за якість відповідають коефіцієнти DWT перетворення у 2, 3 і 4 квадрантах, то маска перетворюється так, щоб вказувати на всі коефіцієнти, що відповідають областям підвищеної якості (рис. 4.20).

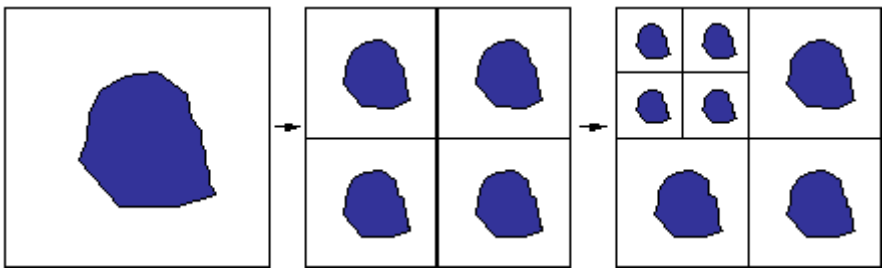


Рисунок 4.20 - Перетворення маски області підвищення якості для обробки DWT коефіцієнтів

Ці області обробляються далі іншими алгоритмами (з меншими втратами), що дозволяє досягти шуканого балансу за якістю і ступенем ущільнення [35, 36].

Характеристики алгоритму JPEG 2000:

Ступінь ущільнення: 2-200 (Задається користувачем). Можливе ущільнення без втрат.

Клас зображень: Повнокольорові 24-бітні зображення. Зображення у градаціях сірого без різких переходів кольорів (фотографії). 1-бітні зображення.

Симетричність: 1-1,5

Характерні риси: Дозволяє видаляти візуально неприємні ефекти, підвищуючи якість у готельних областях. У випадку сильного ущільнення з'являється блочність та великі хвилі у вертикальному та горизонтальному напрямках.

4.6 Відмінності між форматом та алгоритмом

Насамкінець кілька зауважень щодо різниці в термінології, плутанини під час порівняння рейтингів алгоритмів тощо.

Подивіться на короткий перелік форматів, які досить часто використовуються на PC, Apple і UNIX платформах:ADEX, Alpha Microsystems BMP, Autologic, AVHRR, Binary Information File (BIF), Calcomp CCRF, CALS, Core IDC, Cubicomp PictureMaker, Dr. Halo CUT, Encapsulated PostScript, ER Mapper Raster, Erdas LAN/GIS, First Publisher ART, GEM VDI Image File,GIF, GOES, Hitachi Raster Format, PCL, RTL, HP-48sx Graphic Object (GROB), HSIJPEG, HSI Raw, IFF/ILBM, Img Software Set, Jovian VI,JPEG/JFIF, Lumena CEL, Macintosh PICT/PICT2, MacPaint, MTV Ray Tracer Format, OS/2 Bitmap, PCPAINT/Pictor Page Format, PCX, PDS, Portable BitMap (PBM), QDV, QRT Raw, RIX, Scodl, Silicon Graphics Image, SPOT Image, Stork, Sun Icon, Sun Raster, Targa, TIFF, Utah Raster Toolkit Format, VITec, Vivid Format, Windows Bitmap, WordPerfect Graphic File, XBM, XPM, XWD.

В змісті ви можете бачити список алгоритмів компресії. Єдиним збігом виявляється JPEG, а це, погодьтеся, не привід, щоб повсюдно використовувати слова «формат» та «алгоритм компресії» як синоніми (що, на жаль, можна часто спостерігати).

Між цими двома множинами немає взаємно однозначної відповідності. Так, різні модифікації алгоритму RLE реалізовані у величезній кількості форматів - TIFF, BMP, PCX. І, якщо у певному форматі будь-який файл займає багато місця, це не означає, що поганий відповідний алгоритм компресії. Це означає, часто лише те, що реалізація алгоритму, використана у цьому форматі, дає для цього зображення погані результати. Не більше.

Багато сучасних форматів підтримують кілька алгоритмів архівації або не використовують архівації. Наприклад, формат TIFF 6.0 може зберігати зображення з використанням алгоритмів RLE-PackBits, RLE-CCITT, LZW, Гаффмана з фіксованою таблицею, JPEG, а може зберігати зображення без архівації. BMP і TGA дозволяють зберігати файли як з використанням алгоритму компресії RLE (різних модифікацій), так і без його використання.

Висновок 1: Для багатьох форматів, говорячи про розмір файлів, необхідно вказувати, чи використовувався алгоритм компресії і якщо використовувався, то який.

Можна поповнити список ситуацій некоректного порівняння алгоритмів. Під час збереження абсолютно чорного зображення у форматі 1000×1000×256 кольорів у форматі BMP без компресії ми отримуємо, як і належить, файл розміром трохи більше 1000000 байт, а у разі збереження з компресією RLE можна отримати файл розміром 64 байти. Це був би чудовий результат – ущільнення в 15 000 разів (!), якби до нього мала відношення компресія. Справа в тому, що цей файл в 64 байти складається тільки із заголовка зображення, в якому вказано всі його дані. Незважаючи на те, що такий короткий запис зображення став можливим саме завдяки особливості реалізації RLE в BMP, ще раз підкреслимо, що в цьому випадку алгоритм компресії навіть не застосовувався. І те, що для абсолютно чорного зображення 4000×4000×256 ми отримуємо коефіцієнт ущільнення 250 тисяч разів, зовсім не привід для тривалих емоцій щодо ефективності RLE. До речі – цей результат можливий лише за певного стану кольорів на палітрі і далеко не всі програми можуть записувати BMP з архівацією RLE (проте всі стандартні засоби, зокрема засоби системи Windows, читають такий ущільнений файл нормально).

Завжди корисно пам'ятати, що на розмір файлу істотно впливає велика кількість параметрів (варіант реалізації алгоритму, параметри алгоритму (як внутрішні, так і задані користувачем), порядок кольорів на палітрі та багато іншого). Наприклад, для абсолютно чорного зображення 1000×1000×256 градацій сірого у форматі JPEG за допомогою однієї програми за різних параметрів завжди виходив файл приблизно 7 кілобайт. У той самий час, змінюючи опції в іншій програмі, я отримав файли розміром від 4 до 68 Кб (всього на порядок різниці). Водночас декомпресоване зображення для всіх файлів було однаковим – абсолютно чорний квадрат (яскравість 0 для всіх точок зображення).

Справа в тому, що навіть для простих форматів одне і те саме зображення в тому самому форматі з використанням одного і того самого алгоритму архівації можна записати в файл декількома коректними способами. Для складних форматів та алгоритмів архівації виникають ситуації, коли багато програм зберігають зображення різними способами. Така ситуація, наприклад, склалася з форматом TIFF (через його велику гнучкість). Довгий час по-різному зберігали зображення у форматі JPEG, оскільки відповідна група ISO (Міжнародної Організації зі Стандартизації) підготувала лише стандарт алгоритму, але не стандарт формату. Зроблено так було для того, щоби не викликати «війни форматів». Абсолютно протилежне становище зараз із фрактальною компресією, оскільки є стандарт «де-факто» на збереження фрактальних коефіцієнтів у файл

(стандарт формату), але алгоритм їх знаходження (швидкого знаходження!) є технологічною таємницею творців програм-компресорів. Результат – для цілком стандартної програми-декомпресора можуть бути підготовлені файли з коефіцієнтами, що істотно різняться як за розміром, так і за якістю зображення.

Наведені приклади показують, що трапляються ситуації, коли алгоритми запису зображення файл у різних програмах різняться. Однак набагато частіше причиною різниці файлів є різні параметри алгоритму. Як уже говорилося, багато алгоритмів дозволяють у певних межах змінювати свої параметри, але не всі програми дозволяють це робити користувачеві

Висновок 2. Якщо ви не вмієте користуватися програмами архівації або користуєтеся програмами, в яких «для простоти використання» прибрано керування параметрами алгоритму - не дивуйтеся, чому для відмінного алгоритму компресії в результаті виходять великі файли.

4.7 Висновки

Хоча стандарт JPEG 2000 не набув такої популярності як стандарт JPEG, вивчення принципів ущільнення зображень методом JPEG 2000 студентами спеціальності 121 «Інженерія програмного забезпечення» є актуальним.

По-перше, з технічного погляду формат JPEG 2000 забезпечує кращі результати порівняно з JPEG. JPEG 2000 може обробляти та ущільнювати зображення на 200 % краще, ніж JPEG.

По-друге, вивчення принципів роботи JPEG 2000 розширює діапазон знань студентів про ущільнення зображень, показує приклади застосування в ущільненні зображень не тільки ортогональних перетворень, але і дискретного вейвлет-перетворення. Безумовно, що нові стандарти майбутнього будуть містити в собі наробки JPEG 2000.

Щодо чисто технічних переваг і недоліків JPEG 2000, то можна зробити такі висновки. Переваги JPEG 2000 такі:

- найкраща якість зображення за сильного ступеня ущільнення;
- підтримка кодування окремих областей із найкращою якістю;
- підтримка ущільнення без втрат;
- підтримка ущільнення однобітних (2-колірних) зображень;
- на рівні формату підтримується канал прозорості, дозволяє задавати плавний перехід від непрозорого зображення до прозорого фону;
- дає можливість зберігати картинки з прив'язкою до геолокації.

Недоліки JPEG 2000 такі:

- немає зворотної сумісності з оригінальним форматом JPEG, тому користувачам необхідно мати можливість кодувати за стандартом, відмінним від того, який використовують більшість зображень;
- кодеки JPEG 2000 використовують значну кількість пам'яті комп'ютера, що може призвести до зниження продуктивності та часу ущільнення.

4.8 Запитання і вправи для самоконтролю

1. Охарактеризуйте Wavelet-кодування.
2. Наведіть алгоритм кодування зображень методом Wavelet.
3. Які недоліки Wavelet-кодування?
4. Чи ущільнює дані вайвлет-перетворення?
5. Перетворити рядок із 8 байт:

   1, 2, 3, 4, 5, 6, 7, 8.

   Дробові значення округлити. Низькочастотні коефіцієнти розмістити на початку рядка. Ущільнити отриману послідовність методом Гаффмана [24].
6. Послідовність, отриману у попередньому завданні ущільнити арифметичним методом. Порівняти з методом Гаффмана.
7. Виконати обернене перетворення послідовності, отриманої під час виконання завдання 5.
8. Записати загальні вирази для парних і непарних позицій wavelet перетворення як різницеві рівняння низькочастотних і високочастотних фільтрів [37].
9. Для послідовності, отриманої під час виконання завдання 5, виконати повторне перетворення для низькочастотних коефіцієнтів. Отримані результати ущільнити методом Гаффмана і порівняти результати з завданням 5.
10. Послідовність, отриману під час виконання перетворення у завданні 9, ущільнити арифметичним методом, порівняти з результатами завдання 9.
11. Виконати двовимірне перетворення фрагменту 2×2 для $a_{2i,2j=}10$, $a_{2i+1,2j=}10$, $a_{2i,2j+1=}20$, $a_{2i+1,2j+1=}20$.
12. У чому різниця між алгоритмами із втратою інформації та без втрати інформації?
13. Наведіть приклади мір втрати інформації та опишіть їх недоліки.
14. За рахунок чого ущільнює зображення алгоритм JPEG?
15. Чим алгоритм JPEG відрізняється від JPEG 2000?
16. У чому полягає ідея фрактального алгоритму і його відмінності від JPEG?
17. Наведіть основні відмінності дискретного косинусного перетворення від дискретного вайвлет-перетворення.

18. Порівняйте відомі алгоритми ущільнення зображень з JPEG 2000.
19. Для чого виконується зсув за яскравістю?
20. У чому полягає ідея рекурсивного (хвильового) ущільнення?
21. Чи можна застосовувати колірні простори, відмінні від YUV, в алгоритмі JPEG?
22. Як виконується підтримка кодування окремих областей зображення із найкращою якістю в JPEG 2000?
23. Яка основна ідея JPEG 2000?
24. Які відмінності перетворення зображення з колірного простору RGB в колірний простір YUV в JPEG і JPEG 2000?
25. Які види DWT передбачає JPEG 200?
26. За рахунок якого перетворення формуються парні відліки у разі прямого DWT?
27. За рахунок якого перетворення формуються непарні відліки у разі прямого DWT?
28. За рахунок якого перетворення формуються парні відліки під час оберненого DWT?
29. За рахунок якого перетворення формуються непарні відліки під час оберненого DWT?
30. Для чого виконується симетричне розширення зображення?
31. Для чого застосовується квантування в JPEG 2000?
32. Як виконується квантування в JPEG 2000?
33. Яке кодування використовується в JPEG 2000 на етапі ущільнення без втрат?
34. Для чого використовується однобітна маска в JPEG 2000?
35. Для чого використовується перетворення однобітної маски в JPEG 2000?
36. Чи є поняття «формат файлу» і «алгоритм компресії» синонімами?
37. Чим відрізняється формат файлу від алгоритму компресії?
38. Яке розширення мають файли JPEG-2000?
39. Чи підтримують сучасні формати графічних файлів декілька алгоритмів компресії?
40. Чи можуть графічні файли не підтримувати компресію зображень? Наведіть приклади.
41. Скориставшись знаннями з теорії фільтрів [37, 38] запишіть рівняння перетворень JPEG 2000 як різницеві рівняння нерекурсивного фільтру з симетричними коефіцієнтами.
42. Виконайте симетричне розширення послідовності з 8 байт 1, 2, 3, 4, 5, 6, 7, 8, 9 для випадку ущільнення без втрат.
43. Виконайте симетричне розширення послідовності з 8 байт 1, 2, 3, 4, 5, 6, 7, 8, 9 для випадку ущільнення з втратами.
44. Застосуйте пряме та зворотне DWT-перетворення до ланцюжка з 10 байт для випадку ущільнення без втрат: 121, 107, 98, 102, 145, 182, 169, 174, 157, 155.

45. Застосуйте пряме та обернене DWT-перетворення до ланцюжка з 10 байт для випадку ущільнення з втратами: 121, 107, 98, 102, 145, 182, 169, 174, 157, 155.
46. Застосуйте пряме DWT-перетворення під час ущільнення без втрат до ланцюжка з 8 байт: 1, 2, 3, 4, 5, 6, 7, 8. Низькочастотні коефіцієнти розмістить на початку списку й ущільніть методом Гаффмана [24] початкову послідовність і послідовність отриману після перетворення. Порівняйте результати.
47. Початкові дані і дані, отримані в завданні 31 після DWT-перетворення і переупорядкування, ущільніть арифметичним методом [10]. Порівняйте результати.
48. Застосуйте повторно DWT-перетворення до низькочастотних коефіцієнтів DWT-перетворення завдання 31. Ущільніть отримані дані методом Гаффмана і поріняйте з результатами завдання 31.
49. Застосуйте повторно DWT-перетворення до низькочастотних коефіцієнтів DWT-перетворення завдання 31. Ущільніть отримані дані арифметичним методом і порівняйте з результатами завдання 32.
50. Виконайте для випадку ущільнення без втрат пряме і обернене двовимірне DWT-перетворення такого масиву байт:

10,10,10,10

10,10,10,10

10,10,10,10

5 ПРИКЛАД ВИКОРИСТАННЯ НЕОРТОГОНАЛЬНИХ ПЕРЕТВОРЕНЬ ПІД ЧАС УЩІЛЬНЕННЯ ЗОБРАЖЕНЬ

Методи кодування зображень на основі ортогональних перетворень поки що мають найкращі реально досягнуті характеристики за сукупністю таких параметрів, як коефіцієнт ущільнення, якість відновленого після кодування зображення, швидкодія і підтримуються основними виробниками комп'ютерної техніки та техніки зв'язку. Хоча і мають ряд недоліків, зокрема, недостатня достовірність ущільнення складних зображень.

Вирішенням багатьох проблем ортогональних перетворень могло бути застосування неортогональних перетворень, зокрема, методів апроксимації, які дозволяють більш точно відтворити ті чи інші особливості сигналу (зображення). Однак застосування неортогональних перетворень під час ущільнення зображень потребує додаткових досліджень.

При неортогональних базисних функціях використовуються, переважно, степеневі алгебраїчні поліноми типу:

$$s'(t) = \sum_{n=0}^{N-1} c_n t^n, \qquad (5.1)$$

де N – кількість відліків числової послідовності.

Для ущільнення зображень перевага віддається апроксимувальним поліномам, оскільки при цьому видаляються шумові значення сигналу.

5.1 Розробка алгоритму та схеми ущільнення зображень на основі двовимірної апроксимації

У випадку лінійної апроксимації значення піксела для двовимірного зображення визначається так [38]:

$$f(x, y) = ax + by + c. \qquad (5.2)$$

У загальному випадку значення $f(x,y)$ відрізняються від значення піксела $\mathbf{z_{xy}}$. Мінімальне значення відстані досягається за мінімального значення суми квадратів відстаней, тобто:

$$S = \sum_{x=1}^{N} \sum_{y=1}^{M} (ax + by + c - z_{xy})^2 = \mathrm{Min}, \qquad (5.3)$$

де M, N – розміри зображення,

z_{xy} – значення піксела в точці зображення з координатами x, y.

Функція S має мінімальний екстремум у точці, де частинні похідні від коефіцієнтів дорівнюють нулю:

$$\frac{\partial S}{\partial a} = 0, \ \frac{\partial S}{\partial b} = 0, \ \frac{\partial S}{\partial c} = 0. \qquad (5.4)$$

Таким чином, отримаємо систему з трьох рівнянь для трьох невідомих. Для блоків з розміром $n = 4$ система рівнянь така:

$$\begin{cases} 120a + 100b + 40c = \sum_{y=1}^{4}\sum_{x=1}^{4} z_{xy}x \\ 100a + 120b + 40c = \sum_{y=1}^{4}\sum_{x=1}^{4} z_{xy}y \\ 40a + 40b + 16c = \sum_{y=1}^{4}\sum_{x=1}^{4} z_{xy} \end{cases} \quad (5.5)$$

Розв'язавши систему рівнянь (5.5) можна визначити коефіцієнти апроксимації *a, b, c*:

$$b = \frac{-3\sum_{x=1}^{4}\sum_{y=1}^{4} z_{xy} + 1{,}2\sum_{x=1}^{4}\sum_{y=1}^{4} z_{xy}y}{24}, \quad (5.6)$$

$$c = \frac{3\sum_{x=1}^{4}\sum_{y=1}^{4} z_{xy} - \sum_{x=1}^{4}\sum_{y=1}^{4} z_{xy}x - 20b}{8}, \quad (5.7)$$

$$a = \frac{3\sum_{x=1}^{4}\sum_{y=1}^{4} z_{xy} - \sum_{x=1}^{4}\sum_{y=1}^{4} z_{xy}y - 8c}{20}. \quad (5.8)$$

Схема ущільнення зображення на основі двовимірної апроксимації наведено на рис. 5.1.

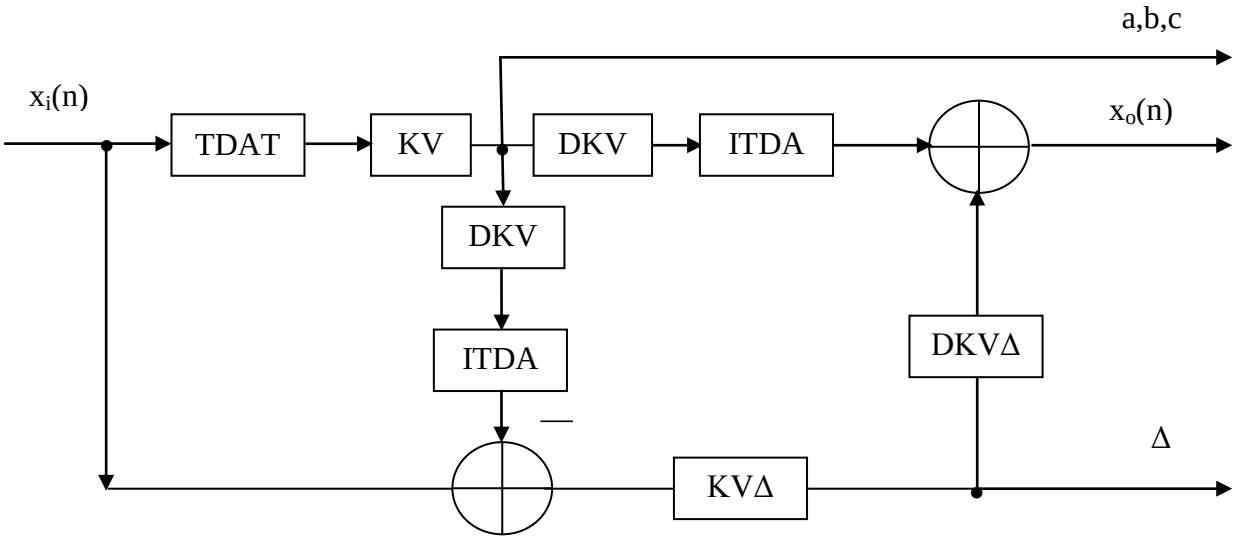


Рисунок 5.1 – Схема ущільнення

У цій схемі призначення основних модулів таке:

- $x_i(n)$ – відліки початкового зображення;

- $x_0(n)$ – відліки відновленого зображення;
- TDAT – двовимірне апроксимувальне перетворення згідно з виразами (3.5 – 3.7);
- ITDAT – обернене апроксимуальне перетворення;
- *a, b, c* – коефіцієнти перетворення;
- $\Delta = x_i(n) - x_0(n)$ – квантований різницевий сигнал;
- KVK, DKVK – квантування, деквантування коефіцієнтів перетворення;
- KVΔ, DKVΔ – квантування, деквантування різницевого сигналу.

Коефіцієнти *a, b, c* та значення Δ ущільнюються в подальшому арифметичним кодером.

Блок-схема алгоритму виконання двовимірного апроксимуючого перетворення приведена на рис. 5.2.

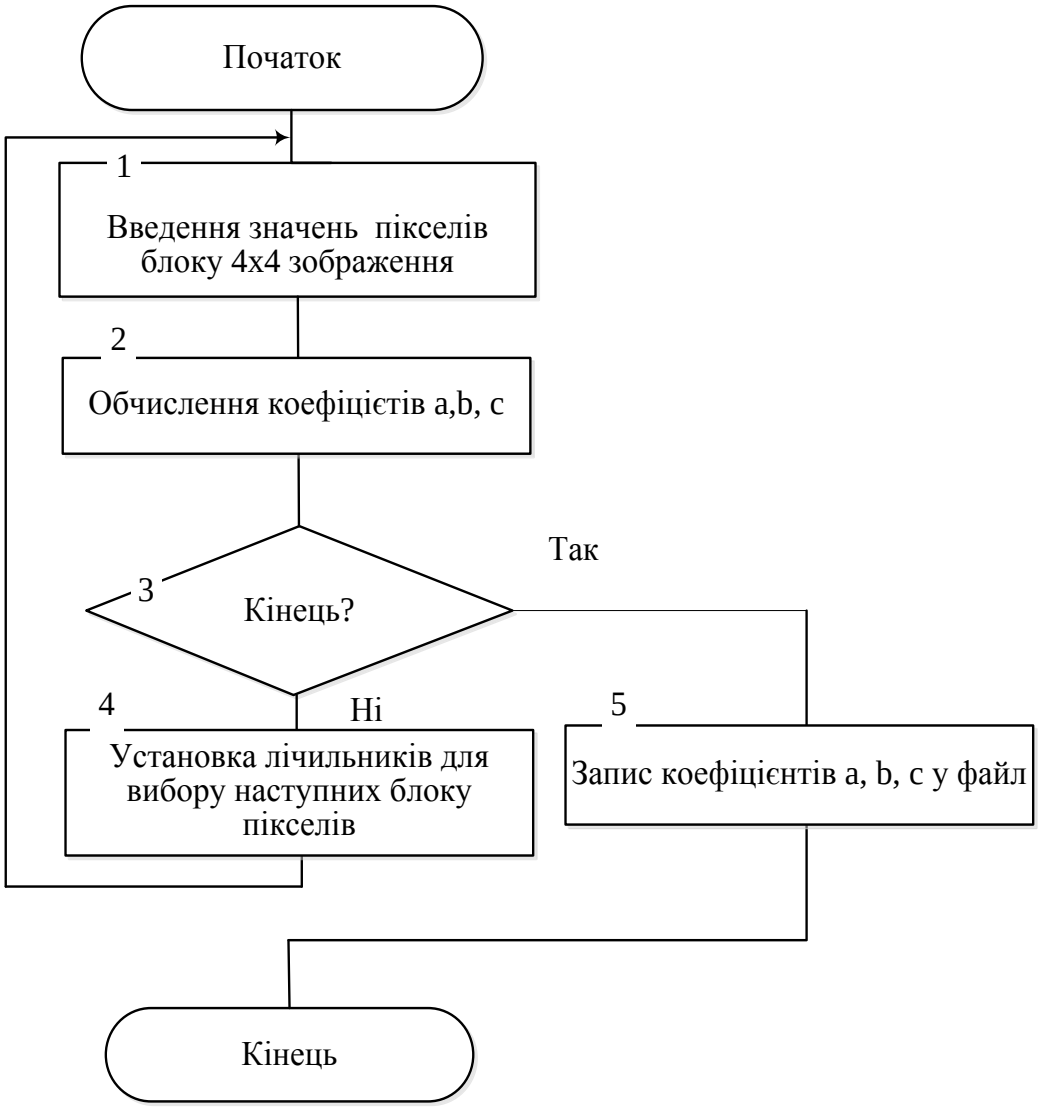


Рисунок 5.2 – Блок-схема виконання апроксимуючого перетворення

Вводяться значення блока пікселів 4×4 зображення (оператор 1) для яких обчислюються значення коефіцієнтів *a, b, c* згідно з виразами 5.6 – 5.8 (оператор 2). Після перевірки на кінець зображення (оператор 3) лічильники установлюються на початок наступного блока з 4×4 пікселів (оператор 4), якщо є ще необроблені групи пікселів, інакше значення коефіцієнтів записуються у файл (оператор 5).

Відновлення зображення для кожної групи пікселів з *N* елементів виконується згідно з таким виразом:

$$f(x_n, y_n) = ax_n + by_n + c$$
$$x_n = 1,2,3,4; y_n = 1,2,3,4 \qquad (5.9)$$

Для кольорового зображення пряме і обернене перетворення потрібно виконувати по кожній компоненті (R,G,B або Y,U,V) окремо.

Ущільнення досягається за рахунок того, що коефіцієнтів перетворення або менше кількості елементів *N* в групі пікселів або їх загальна розрядність менша кількості розрядів в групі з *N* пікселів.

5.2 Результати дослідження

Задача відновлення зображення після кодування полягає в тому, щоб одержати вихідне зображення, яке буде найменш відрізнятись від вихідного зображення.

Для виконання дослідження було розроблено програмне забезпечення, яке в точності відтворювало алгоритм наведений вище. Коефіцієнти a, b, c представлялись 16 бітними значеннями.

Результати, отримані в процесі роботи програми під час кодування зображення, розміром 512×512 з глибиною 24 біт/піксел, продемонстровано на рис. 5.3.

На зображенні (рис. 5.3, В), яке відновлене після неортогонального перетворення, спотворення майже непомітні. Враховуючи схему ущільнення (рис. 5.1), яка передбачає формування різницевого зображення, можна вважати, що якість низькочастотної компоненти зображення (а саме вона формується за неортогональної апроксимації) ідеальна.

Для ущільнення без втрат файлів коефіцієнтів з розширенням .tda будемо використовувати арифметичний кодер dmc.exe. Результати наведено в табл. 5.1.

(A)

(B)

Рисунок 5. 3 – Початкове (А) та відновлене (В) зображення

Таблиця 5.1 – Тестування програми на прикладі файлів Lena

Файл та тип ущільнення	Тип зображення	Початковий розмір даних зображення, байт	Розмір після ущільнення, байт	Час виконання, с	Погіршення візуальної якості
Lena512.bmp (TDA)	напівтонове	786486	61801	<2	Не помітно
Lena512.bmp (Jpeg)	напівтонове	786486	82912	<2	Не помітно
lena512color.tiff (TDA)	кольорове	786572	82365	<2	Не помітно
lena512color.tiff (Jpeg)	кольорове	786572	96387	<2	Не помітно

Аналіз результатів показує, що обсяг зображення зменшується в 3–4 рази. Збіг теоретичних і практичних результатів роботи програми свідчить про її повну роботоздатність.

Завдання для самостійної роботи: виконати програмну реалізацію наведеного алгоритму та провести власні дослідження.

6 ПРИКЛАД ВИКОРИСТАННЯ НЕЙРОННОЇ МЕРЕЖІ ДЛЯ УЩІЛЬНЕННЯ ЗОБРАЖЕНЬ

6.1 Загальна характеристика застосування нейронних мереж до ущільнення зображень

Останні роки характеризуються зростанням інтересу до застосування штучних нейронних мереж для ущільнення зображень [14-17, 39-40].

Дослідження застосування нейронних мереж для ущільнення зображень виконуються в таких напрямках [14-17, 39-40]:

- ущільнення за допомогою нейронних мереж зворотного поширення;
- ущільнення за допомогою алгоритму навчання Хебба;
- застосування нейронних мереж для векторного квантування зображень;
- ущільнення за допомогою нейронних мереж з кодуванням за передбаченням.

Нейронні мережі з кодуванням за передбаченням хоча і широко застосовуються в процесі кодування звукових сигналів, для кодування зображень використовуються мало, тому в подальшому не розглядаються.

Нейронна мережа зворотного поширення містить вхідний, вихідний шари й внутрішній прихований шар. Ущільнення досягається за рахунок того, що прихований шар містить менше нейронів, ніж вхідний і вихідний шари.

Головною перевагою алгоритму Хебба є те, що це алгоритм навчання без учителя. Мережа містить лише два шари, що також є перевагою порівняно з попереднім підходом.

Але найбільш цікавим є інший алгоритм навчання без учителя – алгоритм Кохонена. Мережа Кохонена містить два шари і є ідеальною для вирішення завдань векторного квантування зображень. Алгоритм навчання простіший порівняно з алгоритмом Хебба.

Мережа Кохонена може розпізнавати кластери в даних, а також встановлювати близькість класів. Якщо в даних розпізнані класи, то їх можна позначити, після чого мережа зможе вирішувати завдання класифікації. Мережу Кохонена можна використати й у тих завданнях класифікації, де класи вже задані, тоді перевага буде в тому, що мережа зможе виявити подібність між різними класами.

Мережа Кохонена має всього два шари: вхідний і вихідний, складений із радіальних елементів. Вихідний шар називають також шаром топологічної карти. Елементи топологічної карти розташовуються в деякому просторі, як правило, двовимірному.

Навчається мережа Кохонена методом послідовних наближень. Починаючи з випадково вибраного вихідного розташування центрів,

алгоритм поступово поліпшує його так, щоб установлювати кластеризацію навчальних даних.

Внаслідок ітеративної процедури навчання мережа організується таким чином, що елементи, які відповідають центрам, розташованим близько один до одного в просторі входів, будуть розташовуватися близько один до одного й на топологічній карті. Топологічний шар мережі можна уявляти собі як двовимірну матрицю, які потрібно так відобразити в N-вимірний простір входів, щоб, за змоги, зберегти вихідну структуру даних. Звичайно ж, за будь-якої спроби подати N-вимірний простір на площині будуть загублені багато деталей; однак, такий прийом іноді корисний, тому що він дозволяє візуалізувати дані, які ніяким іншим способом зрозуміти неможливо.

Під час використання основного алгоритму Кохонена послідовно проходить ряд ітерацій, на кожній ітерації обробляється кожний із навчальних прикладів і потім застосовується такий алгоритм:

- вибрати той нейрон, який розташований ближче всього до вхідного прикладу;
- скорегувати нейрон, що виграв, так, щоб він став більше схожий на цей вхідний приклад, взявши зважену суму колишнього центра нейрона й навчального приклада.

Під час обчислення зваженої суми використовується коефіцієнт швидкості навчання, який поступово зменшується. На кожній новій ітерації корегування стає все більш тонким. Результат – положення центра встановлюється в деякій позиції, що задовільно подає ті спостереження, для яких цей нейрон виявився виграшним.

Властивість топологічної впорядкованості досягається в алгоритмі за допомогою додаткового використання поняття околу. Окіл – це кілька нейронів, які оточують нейрон, що виграв. Подібно швидкості навчання, розмір окóлу зменшується згодом, так що спочатку до нього належить досить велике число нейронів, можливо, майже вся топологічна карта. На останніх етапах окіл стає нульовим, тобто складається тільки із самого нейрона, що виграв. Насправді в алгоритмі Кохонена корегування застосовується не тільки до нейрона, що виграв, але й до всіх нейронів із його поточного околу.

Зменшення швидкості навчання та розміру окóлів приводить до більш тонких розходжень усередині ділянок карти, що зрештою приводить до тонкого настроювання кожного нейрона. Часто навчання навмисно розбивають на дві фази: більш коротку, з великою швидкістю навчання й більшими околами, і більш довгу з малою швидкістю навчання й нульовими або майже нульовими околами.

У загальному випадку можна виділити два підходи до застосування нейронних мереж для ущільнення зображень:

- самостійне застосування нейронних мереж;
- у комбінації з відомими методами.

Виходячи з того, що математичною моделлю зображення є випадковий процес із рівномірним законом розподілу, можна припустити, що другий варіант має переваги, оскільки трансформанти зображення характеризуються значно меншою дисперсією порівняно з початковим зображенням. Це, зі свого боку, дозволить зменшити розміри мережі і відповідно обчислювальні витрати на навчання мережі. Перший варіант становить інтерес як підготовчий етап, що дозволяє оцінити цінність і перспективність структури мережі. Тому актуальним є аналіз методів кодування з метою вибору базового для застосування сумісно з нейронною мережею, а також вибір самої нейтронної мережі, яка б дозволила покращити характеристики заданого методу з мінімальними обчислювальними витратами.

З розглянутих підходів до застосування нейронних мереж для ущільнення зображень найбільші переваги має нейронна мережа типу двовимірна карта Кохонена (SOFM - Self Organizing Feature Map), оскільки вона характеризується сумісністю з принципами векторного квантування зображень [14] і алгоритмом некерованого навчання, що дозволяє найкращим чином оптимізувати вхідні вектори [14]. Карта Кохонена має дві важливі властивості, які використовуються під час стиснення зображень методами векторного квантування. По-перше, вона дуже схожа на інші методи векторного квантування, які застосовують під час стиснення зображень із втратами, а по-друге, близьким кластерам вхідних векторів відповідають близько розташовані нейрони, що збільшує ефективність ущільнення без втрат, яке застосовується на наступному етапі кодування.

Однак, незважаючи на перспективність застосування нейронних мереж до ущільнення зображень, викликає запитання швидкість кодування, оскільки етап навчання потребує доволі тривалого часу, що для більшості задач кодування зображення є неприйнятним.

6.2 Схема і алгоритм ущільнення зображень з використанням карти Кохонена

Схему ущільнення зображень з використанням карти Кохонена наведено на рис. 6.1. Після векторизації (перетворення блоків зображення на вектори), виконується векторне квантування з застосуванням карти Кохонена. Вихідні дані векторного квантувача надходять на арифметичний кодер, який виконує кодування зображення без втрат. Декодування виконується в зворотному порядку.

В вихідний файл крім квантованих значень вхідних векторів записується і кодова книга. Але її розмір незначний порівняно з вхідним зображенням і це не впливає на коефіцієнт ущільнення. Векторний квантувач – це карта Кохонена з розміром 16×16 або більшим. Вибір розміру карти Кохонена пояснюється тим, що зображення подається з

точністю 8-біт на елемент зображення для напівтонових зображень, або 24 біти для кольорового зображення.

Застосування арифметичного кодера на етапі ущільнення без втрат забезпечує найбільший коефіцієнт ущільнення порівняно з іншими методами кодування без втрат [4].

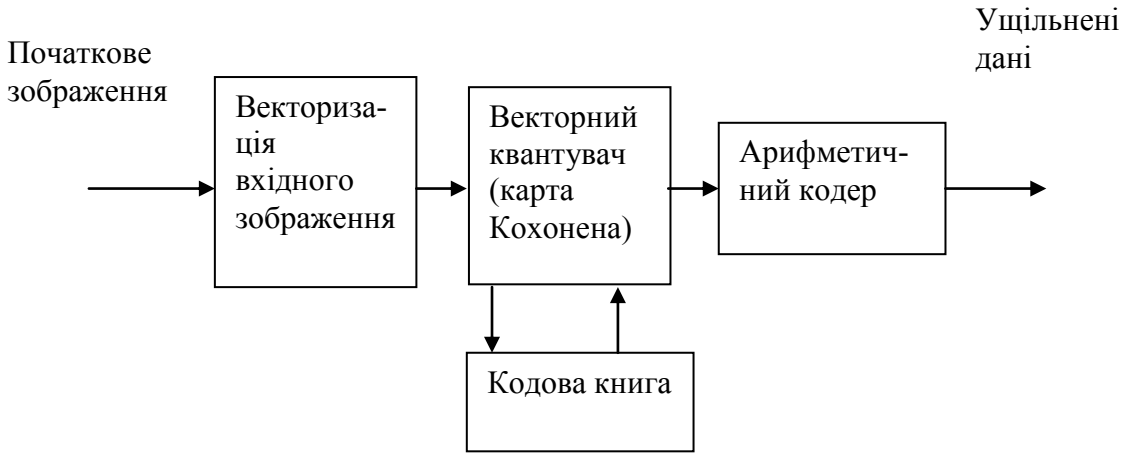


Рисунок 6.1 – Загальна схема кодування (кодер)

Карта ознак Кохонена, що сама організується (Self-Organizing Feature Map – SOFM) має набір вхідних елементів, кількість яких відповідає розмірності вхідних векторів і набір вихідних елементів, які слугують прототипами. Базову архітектуру мережі SOFM наведено на рис. 6.2.

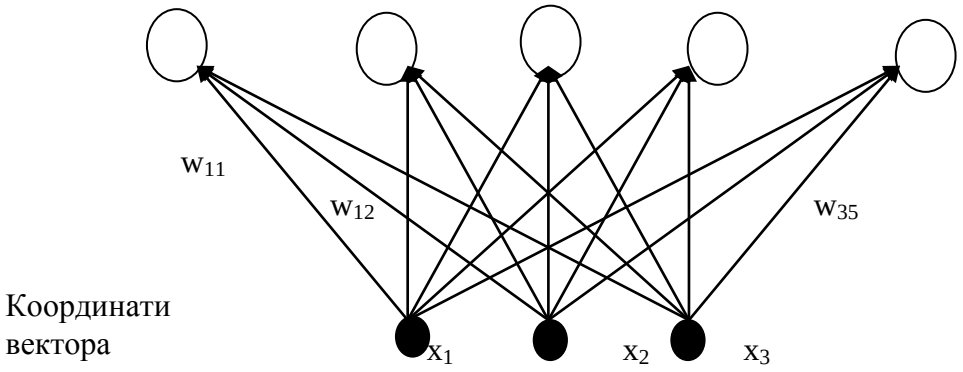


Рисунок 6.2 - Базова архітектура мережі SOFM

Вихідні елементи називаються кластерними елементами. Кластерні елементи або кодові слова розміщуються в вигляді одно- або двовимірного масиву. Звичайно кількість кластерних елементів значно менша порівняно з кількістю навчальних зразків, оскільки метою є отримання спрощеної характеристики вхідних даних. Це і дає можливість використання SOFM як векторного квантувача. Після навчання ця мережа може апроксимувати вектори вхідного простору найкращим способом.

Розмірність вхідних векторів вибирається з урахуванням кореляції вхідних даних. Відомо, що найбільшими кореляційними зв’язками характеризуються сусідні відліки зображення, які можуть мати близькі

значення [4]. Тому зображення розбивається на примикаючі квадрати розміром 2×2, кожний з яких розглядається як вектор в 4-вимірному просторі.

Розмір карти Кохонена визначається мінімальною кількістю розрядів, яка подає елемент зображення і забезпечує достатню якість зображення. З наукової літератури відомо, що менше 2 біт на елемент зображення за будь-яких перетворень досягти проблематично для заданих високих вимог до якості зображення. До того ж, збільшення розміру карти призводить до значної втрати швидкодії навчання мережі. Швидкість навчання прямо пропорційна квадрату розміру сторони карти, тобто:

$$T_н = kN^2,$$

де N – розмір сторони карти,

k – коефіцієнт пропорційності, залежить від конкретної реалізації.

З урахуванням сказаного вище розмір карти Кохонена, який задовольняє ці суперечливі вимоги, вибираємо 16×16. Оскільки розмір вхідного вектора дорівнює чотирьом, така карта забезпечує за векторного квантування таку кількість бітів на елемент зображення:

$$M = \log_2 N^2 / n = 8/4 = 2 \text{ біти/ел.}$$

Що є прийнятним як з погляду швидкодії, так і з погляду забезпечення високої якості відновленого зображення.

Векторне квантування з використанням карти Кохонена виконується за два проходи початкового зображення:

- перший прохід – навчання мережі;
- другий прохід – векторне квантування.

Причому навчальними векторами є всі фрагменти зображення з розмірами 2×2.

Таким чином алгоритм кодування буде таким:

Етап навчання

1. Ініціалізувати вагові нейронів коефіцієнти випадковими значеннями.
2. Вибрати з зображення перший фрагмент 2×2
3. Подати його у вигляді навчального вектора.
4. Для кожного кластерного елемента карти обчислити відстань до навчального вектора:

$$d_j = \sum_{i=0}^{3} (w_{ij} - x_i)^2$$

Знайти кластерний елемент j для якого d_j буде мінімальним.

5. Для цього кластерного елемента обновити вагові коефіцієнти згідно з формулою:

$$w_{ij}(n+1) = w_{ij}(n) + \eta(n)[x_i - w_{ij}(n)],$$

де η – норма навчання,

x_i – координата навчального вектора.

6. Обновити норму навчання η і вибрати з зображення наступний фрагмент 2×2 та повторяти пункти 3 – 6 для наступних навчальних векторів до тих пір, поки не будуть вибрані всі фрагменти.

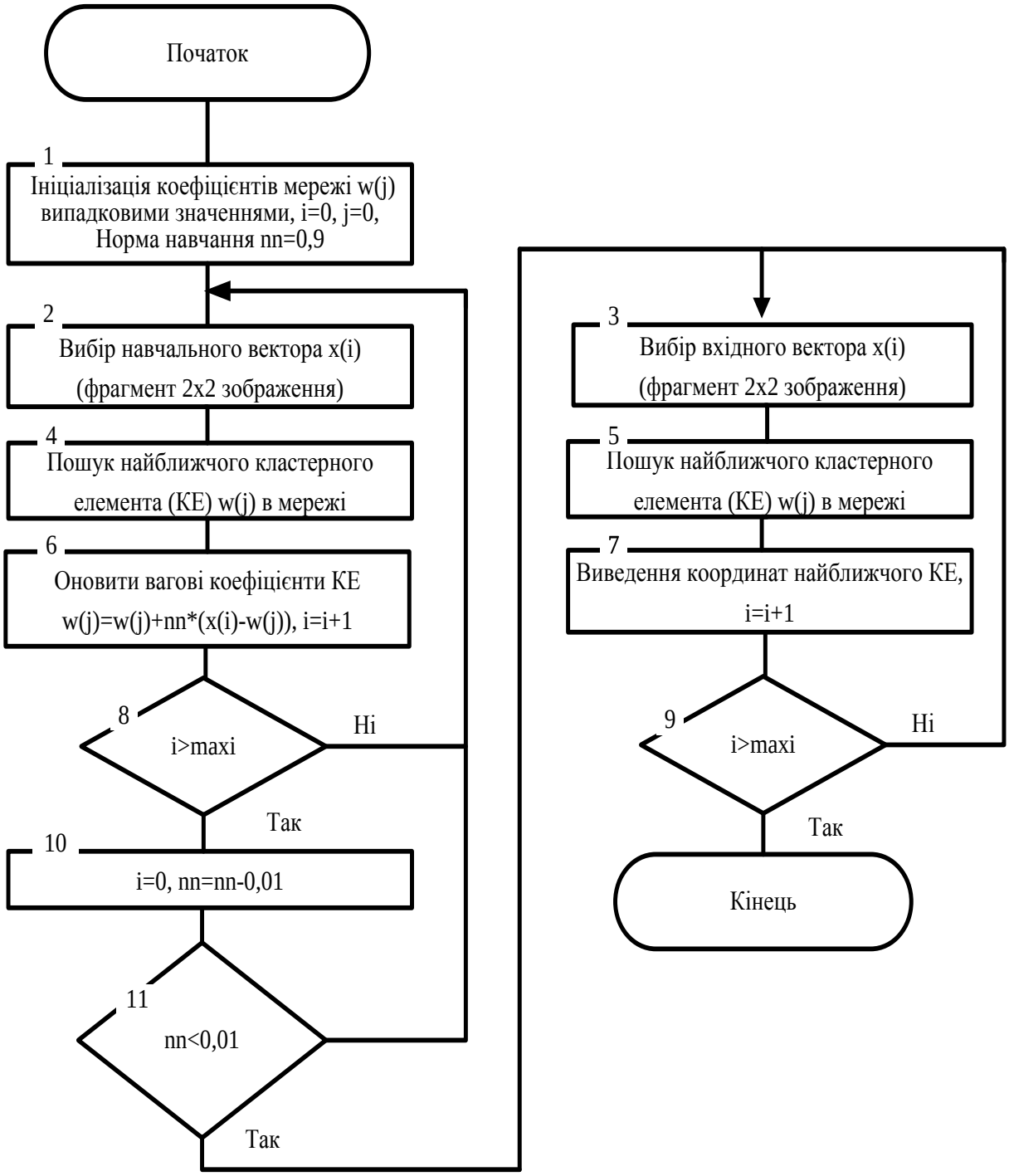


Рисунок 6.3 – Блок-схема алгоритму векторного квантування

Векторне квантування

7. Вибрати з зображення перший фрагмент 2×2.
8. Подати його у вигляді навчального вектора.

9. Для кожного кластерного елемента карти обчислити відстань до навчального вектора:

$$d_j = \sum_{i=0}^{3} (w_{ij} - x_i)^2 .$$

10. Номер кластерного елемента з мінімальним d_j записати в вихідний файл.

Вибрати з зображення наступний фрагмент 2×2 і повторяти пункти 8–11 до тих пір, поки не будуть вибрані всі фрагменти.

11. Записати у вихідний файл значення коефіцієнтів w_{ij} – всього 2048 байт.

12. Записати у вихідний файл розмір початкового файлу.

13. Ущільнити отриманий файл методом арифметичного кодування.

Декодування виконується значно швидше і містить такі етапи:

1. Декодувати ущільнений файл арифметичним методом.
2. Прочитати з вхідного файлу і записати в масив значення номерів кластерних елементів, коефіцієнтів w_{ij} і розмір початкового зображення.
3. Вибрати з масиву номер кластерного елемента для першого фрагмента 2×2.
4. Коефіцієнти w_{ij} цього кластерного елемента (4 коефіцієнти) записати в вихідне зображення як значення елементів відповідного фрагмента 2×2.
5. Вибрати наступний номер кластерного елемента і повторяти пункти 4–5 до тих пір, поки не будуть відновлені всі фрагменти зображення (див. рис. 6.3)

6.3 Результати досліджень

Розглянемо лише застосування векторного квантування безпосередньо до початкового зображення.

Вхідним для модуля векторного квантування є напівтонове або кольорове зображення, подане у форматі 512×512 з глибиною 24 Біт/піксел.

Параметри векторного кантувача такі:

- двовимірна карта Кохонена розміром 16×16 кластерних елементів;
- формат вхідного вектора – двовимірний вектор розміром 2×2.

Очікувані результати роботи програми:

- файл з квантованими компонентами зображення з розширенням .vim;
- ущільнений арифметичним методом файл .vim з розширенням .vimc.

Розмір файлу .vim дорівнює сумі розміру карти Кохонена і векторно квантованих компонент зображення:

$$16\times16\times4\times2+(512/2)\cdot(512/2)\cdot1\cdot3 = 198\ 656 \text{ байт}.$$

Для перевірки правильності роботи цього модуля виберемо файл кольорового зображення стандартної заставки Windows (рис. 6.4). Результати такі:

- файл .vim – має розмір 198 656 байт, що відповідає сподіваним результатам;
- ущільнений файл .vimc має розмір 84 802 байт, СКВ=6, візуальна якість зображення відмінна. Для порівняння розмір цього файлу у форматі ‚jpg 88 041 байт СКВ=5.

Тобто результати порівнянні з форматом ‚jpg.

Однак дослідження виконані для інших зображень показали, що розміру карти Кохонена 16×16 недостатньо для отримання високоякісного відновленого зображення. Тому потрібно збільшити розмір карти Кохонена, але збільшення розміру карти Кохонена зменшує коефіцієнт ущільнення, оскільки навчена карта Кохонена також записується у вихідний файл. Наприклад, для карти Кохонена 32×32 розмір пам’яті для зберігання налаштувань векторного кантувача становитиме:

$$32\cdot32\cdot4\cdot2=8192 \text{ байти}.$$

З одної сторони, це не так і багато за ущільнення файлів розміром 1 Мбайт і більше, але з іншої сторони для вказання позиції найближчого кластерного елемента уже потрібно 10 біт, що збільшує файл квантованих компонент зображення в 1,5 раза і відповідно зменшується коефіцієнт ущільнення зображення.

Рисунок 6.4 – Заставка Windows, СКВ=6

Завдання для самостійної роботи: виконати програмну реалізацію наведеного алгоритму та провести власні дослідження.

7 ПРОГРАМНА ІНЖЕНЕРІЯ ДЛЯ СИСТЕМ КОДУВАННЯ ЗОБРАЖЕНЬ

7.1 Мови програмування та бібліотеки

Для розробки систем кодування та стиснення зображень використовуються різні мови програмування та бібліотеки.

Python. Дуже популярний для швидкого прототипування, аналізу даних та застосувань машинного навчання в обробці зображень [41, 42, 43].

- **OpenCV (Open Computer Vision Library).** Велика бібліотека з відкритим вихідним кодом для комп'ютерного зору, машинного навчання та обробки зображень. Має широку підтримку Python та C++.
- **Pillow (PIL).** Зручна бібліотека Python для базових операцій із зображеннями, підтримки форматів файлів та ефективного внутрішнього подання.
- **SciPy.** Корисна для математичних задач та процедур в обробці зображень.
- **Scikit-image** Модуль Python, що пропонує багату колекцію алгоритмів для обробки зображень.
- **ImageIO.** Спрощує читання та записування зображень у різних форматах.
- **Matplotlib.** Важливий для відображення зображень та візуалізації.
- **Фреймворки глибокого навчання.** TensorFlow, PyTorch (необхідні для дослідження стиснення зображень на основі глибокого навчання).
- **C++.** Кращий для застосувань, критичних до продуктивності, систем реального часу та низькорівневого контролю [44].
- **OpenCV.** Надає надійний C++ API.
- **CImg Library.** Мала та відкрита бібліотека C++ для обробки зображень, відома своєю самодостатністю, потокобезпечністю та високою портативністю.
- **PXIPL.** Бібліотека C/C++ для обробки та аналізу зображень із широкою сумісністю.
- **GLM (OpenGL Mathematics).** Математична бібліотека C++, корисна для графічного програмного забезпечення.

Інструменти обробки відео

- **FFmpeg.** Потужний інструмент командного рядка та бібліотека з відкритим вихідним кодом для обробки мультимедійних даних, включаючи кодування, декодування, транскодування та потокову передачу відео та аудіо. Підтримує широкий спектр кодеків, включно з HEVC [45].

Інтегровані середовища розробки (IDE):

- Для Python: PyCharm, Visual Studio Code (VS Code), Jupyter, Spyder.
- Для C++: Visual Studio Code.

Таблиця 7.1 – Рекомендовані програмні бібліотеки та інструменти для кодування зображень

Категорія	Конкретні інструменти/ бібліотеки	Ключові функції/переваги	Основні сценарії використання
Мови програмування	Python , C++	Швидке прототипування, багаті бібліотеки (Python); продуктивність, низькорівневий контроль (C++)	Дослідження, розробка прототипів (Python); виробничі системи, реальний час (C++)
Основні бібліотеки обробки зображень	OpenCV, Pillow (PIL), SciPy, Scikit-image, CImg	Комплексні алгоритми, підтримка форматів, маніпуляції, фільтри, морфологія	Загальна обробка зображень, комп'ютерний зір, наукові обчислення
Мультимедійні фреймворки	FFmpeg	Кодування/ декодування відео та зображень, конвертація форматів	Обробка відео, інтеграція мультимедіа в додатки
IDE	PyCharm, VS Code, Jupyter	Інтелектуальне автодоповнення, налагодження, інтеграція VCS	Розробка, дослідження, аналіз даних
Інструменти контролю версій	Git/Git LFS, Adobe Lightroom/Apertur, git-annex	Відстеження змін, спільна розробка, управління великими бінарними файлами	Управління кодом та активами зображень у проектах

7.2 Принципи архітектури програмного забезпечення

Застосування основних принципів програмної інженерії є вирішальним для побудови надійних, підтримуваних та масштабованих систем кодування зображень.

- **Розділення відповідальності (SoC).** Розділення програмної системи на менші, незалежні компоненти, кожен з яких має одну, чітко визначену відповідальність. Для обробки зображень це означає розділення збору зображень, конвеєрів обробки (наприклад,

перетворення, квантування, ентропійне кодування), оцінення якості та користувацького інтерфейсу на окремі модулі.

- **Абстракція.** Відображення лише основних функцій програмного компонента, приховуючи деталі його внутрішньої реалізації.
- **Модульність.** Розбиття системи на менші, незалежні модулі, кожен з яких виконує певну функцію з чітким інтерфейсом.[7]
- **Принцип «Не повторюйся» (DRY).** Зменшення повторень у кодовій базі.
- **Принцип «Зроби це просто, дурню» (KISS).** Заохочує до створення простих рішень, що відповідають вимогам без зайвих ускладнень.
- **Висока зв'язність та низька зв'язаність.** Забезпечення того, що елементи в модулі тісно пов'язані (зв'язність) і що модулі мають мінімальні залежності один від одного (зв'язаність).

7.3 Стратегії оптимізації продуктивності

Стиснення та декомпресія зображень можуть бути обчислювально інтенсивними.

- **Апаратне прискорення.** Використання спеціалізованого апаратного забезпечення (наприклад, графічних процесорів) може значно покращити продуктивність.
- **Паралельна обробка.** Впровадження паралельної обробки для підвищення швидкості процесу стиснення, особливо для великих наборів даних зображень.
- **Адаптивне потокове передавання.** Для відео та веб-контенту адаптивне потокове передавання бітрейту дозволяє компенсувати відмінності в інтернет-з'єднаннях кінцевих користувачів шляхом динамічного регулювання якості. Ключовими є протоколи, такі як HTTP Live Streaming (HLS) та MPEG-DASH.
- **Профілювання та політики оптимізації.** Автоматизація профілювання зображень під час процесів збірки та встановлення політик оптимізації за призначенням зображення.
- **Ефективна доставка.** Передача зображень через HTTP/3 та пріоритизація за вікном перегляду.

7.4 Контроль версій та спільна розробка в проєктах, орієнтованих на зображення

Системи контролю версій (VCS) є важливими для відстеження змін та спільної роботи над програмними проєктами. Для обробки зображень це означає підтримку історії кожної зміни, внесеної до зображення.

Проте, існують виклики для традиційних VCS. Стандартні VCS (наприклад, Git) можуть мати труднощі з великими бінарними файлами, такими як зображення, оскільки більшість пікселів змінюються, і економія

місця від зберігання лише змін є незначною. Це призводить до великих розмірів репозиторіїв.

Існують спеціалізовані рішення:

- **Недеструктивний робочий процес:** Додатки для фотографічного робочого процесу, такі як Adobe Lightroom, пропонують недеструктивний робочий процес, де зміни зберігаються як інструкції, а не змінюють оригінальне зображення.
- **Інструменти для великих файлів:** Інструменти, такі як git-annex, можуть керувати великими файлами, фіксуючи символічні посилання, зберігаючи фактичні файли поза основним репозиторієм.
- **Системи управління цифровими активами (DAM):** З можливостями контролю версій рекомендуються для управління активами зображень у різних командах.

7.5 Приклади використання Pillow та OpenCV

1. Python. Бібліотеки Pillow та OpenCV

Pillow – це форк бібліотеки Python Imaging Library (PIL), який використовується для обробки зображень. Щоб встановити Pillow, виконайте команду: pip install Pillow.

OpenCV – це бібліотека з відкритим кодом для обробки зображень і комп’ютерного зору. Вона підтримує операції, такі як фільтрація, трансформація, сегментація та кодування зображень. Для встановлення бібліотеки OpenCV для Python найпростіше скористатися менеджером пакетів pip, відкривши командний рядок або термінал та ввівши команду pip install opencv-python.

Приклади використання Pillow

Приклад 1. Кодування зображення у різні формати

Цей приклад демонструє, як відкрити зображення та зберегти його у форматах JPEG, PNG і WebP із різними параметрами.

```
from PIL import Image
# Відкриття зображення
image = Image.open("input.png")
# Збереження у форматі JPEG (стискання з втратами)
image.save("output.jpg", "JPEG", quality=85)  # Якість від 1 до 100
# Збереження у форматі PNG (стискання без втрат)
image.save("output.png", "PNG", optimize=True)  # Оптимізація для зменшення розміру
# Збереження у форматі WebP
image.save("output.webp", "WEBP", quality=80)  # Якість для WebP з втратами
```

Пояснення:

- quality=85: Контролює рівень стискання для JPEG і WebP (1 — низька якість, 100 — висока).

- optimize=True: Для PNG зменшує розмір файлу без втрати якості.

Приклад 2. Зміна розміру та кодування

Зміна розміру зображення часто використовується для оптимізації перед кодуванням.

```
from PIL import Image
# Відкриття зображення
image = Image.open("input.png")
# Зміна розміру до 300x300 пікселів
resized_image = image.resize((300, 300), Image.LANCZOS)  # LANCZOS для якісного масштабування
 # Збереження у форматі JPEG
resized_image.save("resized_output.jpg", "JPEG", quality=90)
```

Пояснення:

- Image. LANCZOS: Алгоритм масштабування, який забезпечує високу якість.
- Зміна розміру зменшує розмір файлу та прискорює обробку.

Приклад 3. Перетворення зображення з PNG у WebP із прозорістю

WebP підтримує прозорість, подібно до PNG. Цей приклад показує, як зберегти прозорість під час конвертації.

```
from PIL import Image

# Відкриття зображення з прозорістю
image = Image.open("input_with_alpha.png")
# Перетворення у режим RGBA (якщо ще не в цьому режимі)
image_rgba = image.convert("RGBA")
# Збереження у WebP із підтримкою прозорості
image_rgba.save("output_with_alpha.webp", "WEBP", lossless=True)
```

Пояснення:

- lossless=True: Зберігає зображення без втрат, що підтримує прозорість.
- convert("RGBA"): Гарантує, що зображення підтримує альфа-канал.

Приклад 4. Масова обробка зображень

Цей приклад показує, як обробити кілька зображень у папці, змінити їх розмір і зберегти у форматі JPEG.

```
from PIL import Image
import os
# Папка із зображеннями
input_folder = "images"
output_folder = "output_images"
```

```
# Створення вихідної папки, якщо вона не існує
if not os.path.exists(output_folder):
    os.makedirs(output_folder)
# Об Old: Обробка всіх зображень у папці
for filename in os.listdir(input_folder):
    if filename.endswith((".png", ".jpg", ".jpeg")):
        # Відкриття зображення
        image_path = os.path.join(input_folder, filename)
        image = Image.open(image_path)
        # Зміна розміру
        resized_image = image.resize((800, 600), Image.LANCZOS)
        # Збереження у форматі JPEG
        output_path = os.path.join(output_folder, f"processed_{filename.split('.')[0]}.jpg")
        resized_image.save(output_path, "JPEG", quality=85)
```

Пояснення:

- os.listdir: Перебирає файли в папці.
- Перевіряє розширення файлів, щоб обробляти лише зображення.
- Зберігає оброблені зображення у новій папці.

Параметри стискання (Pillow)

Pillow дозволяє налаштовувати параметри стискання:

- **JPEG**: Використовуйте параметр quality (1–100). Менше значення = менший розмір файлу, але гірша якість.
- **PNG**: Параметр optimize=True зменшує розмір файлу без втрат.
- **WebP**: Підтримує quality (для стискання з втратами) або lossless=True (без втрат).

Приклади використання OpenCV

Приклад 5. Зміна яскравості зображення (opencv)

```
import cv2
import numpy as np
def adjust_brightness(input_path, output_path, factor=1.5): img =
    cv2.imread(input_path)
    hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
    h, s, v = cv2.split(hsv)
    v = np.clip(v * factor, 0, 255).astype(np.uint8) hsv =
    cv2.merge([h, s, v])
    result = cv2.cvtColor(hsv, cv2.COLOR_HSV2BGR)
    cv2.imwrite(output_path, result)
adjust_brightness('input.png', 'brightened.png', 1.5)
```

Приклад 6. Кодування зображення у різні формати (opencv)

```
import cv2
from PIL import Image
import sys
```

```
def compress_image(input_path, output_path, format='JPEG', quality=90):
    if format not in ['JPEG', 'PNG', 'WEBP']:
        print("Не підтримуваний формат")
        return
    img = cv2.imread(input_path)
    if img is None:
        print("Помилка: невдалося завантажити зображення ")
        return
    if format == 'JPEG':
        cv2.imwrite(output_path, img,
            [int(cv2.IMWRITE_JPEG_QUALITY), quality])
    elif format == 'PNG':
        cv2.imwrite(output_path, img,
            [int(cv2.IMWRITE_PNG_COMPRESSION), quality // 10])
    elif format == 'WEBP':
        img_pil = Image.fromarray(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        img_pil.save(output_path, 'WEBP', quality=quality)
    print(f"Зображення збереженояк {output_path}")
if __name__ == "__main__":
    compress_image('input.png', 'output.webp', format='WEBP', quality=80)
```

Приклад 7. Обчислення PSNR (opencv)

```
import cv2
import numpy as np
def calculate_psnr(original_path, compressed_path): original =
    cv2.imread(original_path) compressed =
    cv2.imread(compressed_path)
    mse = np.mean((original - compressed) ** 2) if mse
    == 0:
        return float('inf')
    max_pixel = 255.0
    psnr = 20 * np.log10(max_pixel / np.sqrt(mse)) return psnr
psnr = calculate_psnr('input.png', 'output.jpg')
print(f"PSNR: {psnr:.2f} dB")
```

Рекомендації для платформ (Pillow)

Windows

- Встановіть Python із офіційного сайту (python.org).
- Використовуйте pip для встановлення Pillow.
- Переконайтеся, що у вас є достатньо пам'яті для обробки великих зображень.

Linux (наприклад, Ubuntu)

- Встановіть Python і Pillow:
- sudo apt update
- sudo apt install python3 python3-pip

- pip3 install Pillow
- Для WebP може знадобитися бібліотека libwebp:
- sudo apt install libwebp-dev

macOS

- Встановіть Python через Homebrew:
- brew install python
- pip3 install Pillow
- Для підтримки WebP встановіть libwebp:
- brew install webp

2. C++: Бібліотека OpenCV

Опис

OpenCV (Open Source Computer Vision Library) - це потужна бібліотека для обробки зображень і комп'ютерного зору, яка підтримує кодування зображень у формати JPEG, PNG, WebP тощо. Вона особливо корисна для високопродуктивних застосунків, де потрібна швидка обробка великих обсягів даних.

Встановлення OpenCV

Для використання OpenCV на C++ потрібно встановити бібліотеку та налаштувати компілятор.

Linux (Ubuntu)

sudo apt update
sudo apt install libopencv-dev

Windows

1. Завантажте OpenCV із офіційного сайту (opencv.org) або використовуйте менеджер пакетів, наприклад, vcpkg:
2. vcpkg install opencv
3. Налаштуйте шляхи до бібліотек у вашому компіляторі (наприклад, Visual Studio).

macOS

brew install opencv

Приклади використання OpenCV

Приклад 1: Кодування зображення у різні формати

Цей приклад показує, як відкрити зображення та зберегти його у форматах JPEG, PNG і WebP.

```
#include <opencv2/opencv.hpp>
using namespace cv;
int main() {
   // Відкриття зображення
   Mat image = imread("input.png");
   if (image.empty()) {
      std::cerr << "Помилка: не вдалося відкрити зображення!" << std::endl;
```

```
    return -1;
  }
  // Збереження у форматі JPEG (якість 85)
  std::vector<int> jpg_params;
  jpg_params.push_back(IMWRITE_JPEG_QUALITY);
  jpg_params.push_back(85);
  imwrite("output.jpg", image, jpg_params);
  // Збереження у форматі PNG
  std::vector<int> png_params;
  png_params.push_back(IMWRITE_PNG_COMPRESSION);
  png_params.push_back(9); // Рівень стискання від 0 до 9
  imwrite("output.png", image, png_params);
  // Збереження у форматі WebP
  std::vector<int> webp_params;
  webp_params.push_back(IMWRITE_WEBP_QUALITY);
  webp_params.push_back(80); // Якість від 0 до 100
  imwrite("output.webp", image, webp_params);
  return 0;
}
```

Пояснення:

- imread: Завантажує зображення у пам’ять.
- IMWRITE_JPEG_QUALITY, IMWRITE_PNG_COMPRESSION, IMWRITE_WEBP_QUALITY: Параметри для контролю стискання.
- Перевірка image.empty() запобігає помилкам, якщо зображення не завантажилося.

Приклад 2. Зміна розміру та кодування

Цей приклад демонструє зміну розміру зображення перед збереженням у форматі JPEG.

```
#include <opencv2/opencv.hpp>
using namespace cv;
int main() {
  // Відкриття зображення
  Mat image = imread("input.png");
  if (image.empty()) {
    std::cerr << "Помилка: не вдалося відкрити зображення!" << std::endl;
    return -1;
  }
  // Зміна розміру до 300x300
  Mat resized_image;
  resize(image, resized_image, Size(300, 300), 0, 0, INTER_LANCZOS4);
  // Збереження у форматі JPEG
  std::vector<int> jpg_params;
  jpg_params.push_back(IMWRITE_JPEG_QUALITY);
  jpg_params.push_back(90);
  imwrite("resized_output.jpg", resized_image, jpg_params);
  return 0;
```

```
}
```

Пояснення:

- INTER_LANCZOS4: Алгоритм масштабування для високої якості.
- resize: Змінює розмір зображення до вказаних розмірів.

Приклад 3. Перетворення з PNG у WebP із прозорістю

OpenCV підтримує прозорість (альфа-канал) для PNG і WebP.

```
#include <opencv2/opencv.hpp>
using namespace cv;
int main() {
   // Відкриття зображення з прозорістю
   Mat image = imread("input_with_alpha.png", IMREAD_UNCHANGED);
   if (image.empty()) {
      std::cerr << "Помилка: не вдалося відкрити зображення!" << std::endl;
      return -1;
   }
   // Перевірка наявності альфа-каналу
   if (image.channels() != 4) {
      std::cerr << "Зображення не має альфа-каналу!" << std::endl;
      return -1;
   }
   // Збереження у форматі WebP із підтримкою прозорості
   std::vector<int> webp_params;
   webp_params.push_back(IMWRITE_WEBP_QUALITY);
   webp_params.push_back(100); // Без втрат
   imwrite("output_with_alpha.webp", image, webp_params);
   return 0;
}
```

Пояснення:

- IMREAD_UNCHANGED: Завантажує зображення з альфа-каналом.
- image.channels() == 4: Перевіряє наявність альфа-каналу.
- IMWRITE_WEBP_QUALITY із значенням 100 забезпечує стискання без втрат.

Приклад 4. Масова обробка зображень

Цей приклад обробляє всі зображення в папці, змінює їх розмір і зберігає у форматі JPEG.

```
#include <opencv2/opencv.hpp>
#include <filesystem>
#include <vector>
using namespace cv;
using namespace std;
namespace fs = std::filesystem;
int main() {
   string input_folder = "images";
   string output_folder = "output_images";
   // Створення вихідної папки
```

```
    fs::create_directories(output_folder);
    // Перебір файлів у папці
    for (const auto& entry : fs::directory_iterator(input_folder)) {
        string ext = entry.path().extension().string();
        if (ext == ".png" || ext == ".jpg" || ext == ".jpeg") {
            // Відкриття зображення
            Mat image = imread(entry.path().string());
            if (image.empty()) continue;
            // Зміна розміру
            Mat resized_image;
            resize(image, resized_image, Size(800, 600), 0, 0, INTER_LANCZOS4);
            // Збереження у форматі JPEG
            string output_path = output_folder + "/processed_" + entry.path().stem().string() +
".jpg";
            std::vector<int> jpg_params;
            jpg_params.push_back(IMWRITE_JPEG_QUALITY);
            jpg_params.push_back(85);
            imwrite(output_path, resized_image, jpg_params);
        }
    }
    return 0;
}
```

Пояснення:

- std::filesystem: C++ API для роботи з файлами.
- Перевіряє розширення файлів і обробляє лише зображення.
- Зберігає результати в окремій папці.

Параметри стискання (OpenCV)

- **JPEG**: IMWRITE_JPEG_QUALITY (0–100).
- **PNG**: IMWRITE_PNG_COMPRESSION (0–9, де 9 — максимальне стискання).
- **WebP**: IMWRITE_WEBP_QUALITY (0–100, 100 для стискання без втрат).

Рекомендації для платформ (OpenCV)

Linux (Ubuntu)

- Встановіть необхідні залежності:
  sudo apt update
  sudo apt install libopencv-dev
- Налаштуйте компілятор (наприклад, g++):
  g++ -o program program.cpp `pkg-config --cflags --libs opencv4`

Windows

- Використовуйте vcpkg або завантажте OpenCV із офіційного сайту.
- Налаштуйте Visual Studio, додавши шляхи до бібліотек OpenCV у налаштуваннях проєкту.

macOS

- Встановіть OpenCV через Homebrew:

brew install opencv

- Налаштуйте компілятор:
  g++ -o program program.cpp `pkg-config --cflags --libs opencv4`

Порівняння Pillow і OpenCV

Бібліотека	Мова	Формати	Продуктивність	Складність
Pillow	Python	JPEG, PNG, WebP, тощо	Висока	Низька
OpenCV	C++	JPEG, PNG, WebP	Дуже висока	Висока

- **Pillow**: Простіша у використанні, ідеальна для швидкого прототипування та скриптів.
- **OpenCV**: Потужніша, швидша, підходить для високопродуктивних застосунків і комп'ютерного зору.

Висновки

Pillow (Python) і OpenCV (C++) є потужними інструментами для кодування зображень.

Pillow підходить для простих завдань і швидкої розробки, тоді як OpenCV пропонує високу продуктивність і розширені можливості для обробки зображень.

Формат WebP є перспективним для вебдодатків завдяки ефективному стисканню та підтримці прозорості.

7.6 Запитання для самоконтролю

1. Які мови програмування найчастіше використовуються для розробки систем кодування та стиснення зображень?
2. Які переваги мови програмування Python під час використання в системах обробки зображень?
3. Які переваги мови програмування C++ під час використання в системах обробки зображень?
4. Наведіть характеристики основних бібліотек для обробки зображень.
5. Охарактеризуйте принципи програмної інженерії для побудови надійних, підтримуваних та масштабованих систем кодування зображень.
6. Які стратегії оптимізації продуктивності застосовують в програмних системах обробки зображень?
7. Які параметри ущільнення зображень підтримує бібліотека Pillow?
8. Які параметри ущільнення зображень підтримує бібліотека OpenCV?
9. Наведіть порівняльні характеристики бібліотек Pillow і OpenCV.
10. На яких операційних системах можна використовувати бібліотеки Pillow і OpenCV?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

1. Gonzalez R. C., Woods R. E. Digital Image Processing. 4th ed., Pearson, 2018.
2. Sayood K. Introduction to Data Compression. 5th ed., Morgan Kaufmann, 2017.
3. Salomon D. Data Compression: The Complete Reference. 4th ed., Springer, 2007.
4. Майданюк В. П. Методи і засоби комп'ютерних інформаційних технологій. Кодування зображень : навч. посіб. Вінниця: ВДТУ, 2001. 65 с.
5. Basic principles of digital image compression – FutureLearn. URL: https://www.futurelearn.com/info/courses/introduction-to-digital-media/0/steps/428712 (дата звернення: 04.06.2025).
6. Redundancy in digital images. URL: https://www.educative.io/answers/redundancy-in-digital-images (дата звернення: 04.06.2025).
7. Video Compression - What is it and how does it work? URL: https://getstream.io/glossary/video-compression/ (дата звернення: 04.06.2025).
8. Understanding VMAF, PSNR, and SSIM: Full-Reference video quality metrics. URL: https://www.fastpix.io/blog/understanding-vmaf-psnr-and-ssim-full-reference-video-quality-metrics (дата звернення: 04.06.2025).
9. Кавка О., Майданюк В., Романюк О., Завальнюк Є. Аналіз алгоритмів стиснення зображень із втратами. *Інформаційні технології та комп`ютерна інженерія*. 2023. Вип. 58, № 3. С. 59–64.
10. Майданюк В. П., Романюк О. Н., Павлов С. В., Нечипорук М. Л. Підвищення коефіцієнта ущільнення зображень на основі двовимірних ортогональних перетворень. *Наукові праці Донецького національного технічного університету. Серія «Інформатика, кібернетика та обчислювальна техніка»*. Всеукр. наук. зб. Дрогобич : ДонНТУ, 2024. № 2 (39). С. 31–40.
11. Майданюк В. П. Методичні вказівки до самостійної роботи з дисципліни «Теорія та програмне забезпечення цифрової обробки сигналів і зображень» із спеціальності «Інженерія програмного забезпечення». [Електронний ресурс] Вінниця : ВНТУ, 2025. (PDF, 39 с.).
12. Maydaniuk V. P., Arseniuk I. R., Lishchuk O. O. Increasing the Speed of Fractal Image Compression Using Two-Dimensional Approximating Transformations. *JOURNAL OF ENGINEERING SCIENCES*. Web site: http://jes.sumdu.edu.ua, DOI:10.21272/jes.2019.6(1).e3, Volume 6. Issue 1 (2019). P. E16–E20.
13. Майданюк В. П., Ліщук О. О., Король Д. С. Аспекти оптимізації швидкості фрактального ущільнення зображень. *Оптико-електронні інформаційно-енергетичні технології*. 2017. № 1. С. 24–32.
14. Shadi M. S. Hilles, Volodymyr P. Maidaniuk. SOFM for Image Compression Based on Spatial Frequency Band-Pass Filter and Vector Quantization / Handbook of Research on Intelligent Data Processing and

Information Security Systems Stepan Mykolayovych Bilan (State University of Infrastructure and Technology, Ukraine) and Saleem Issa Al-Zoubi (Irbid National University, Jordan) Copyright: © 2020 |Pages: 434.
15. Universal Deep Image Compression via Content-Adaptive Optimization with Adapters. URL: https://arxiv.org/abs/2211.00918 (дата звернення: 04.06.2025).
16. Deep-learning-based image compression for microscopy images. URL: https://pmc.ncbi.nlm.nih.gov/articles/PMC11704128/ (дата звернення: 05.06.2025).
17. Image Compression Based on Deep Learning: A Review – ResearchGate. URL:https://www.researchgate.net/publication/351315921_Image_Compression_Based_on_Deep_Learning_A_Review (дата звернення: 05.06.2025).
18. JPEG Compression step by step. Introduction. URL: https://medium.com/@pasanSK/jpeg-compression-step-by-step-8d84598190c (дата звернення: 05.06.2025).
19. HEIF vs JPEG: 4 Key Differences and How to Choose. URL: https://cloudinary.com/guides/image-formats/heif-vs-jpeg-4-key-differences-and-how-to-choose (дата звернення: 05.06.2025).
20. Image Compression Algorithms: What Developers Need to Know. URL: https://dev.to/hardik_b2d8f0bca/image-compression-algorithms-what-developers-need-to-know-5135 (дата звернення: 05.06.2025).
21. Documentation on JPEG-2000. JPEG-2000. URL: https://jpeg.org/jpeg2000/documentation.html (дата звернення: 05.06.2025).
22. JPEG 2000 Image Compression. URL: https://www.analog.com/en/resources/analog-dialogue/articles/jpeg-2000-image-compression.html (дата звернення: 05.06.2025).
23. Пєтух А. М., Майданюк В. П., Ліщук О. О. Аналіз алгоритмів стиснення даних і їх програмних реалізацій. *Інформаційні технології та комп`ютерна інженерія*. 2016. № 2. С. 4–9.
24. Майданюк В. П., Романюк О. Н., Тужанський С. Є. Основи теорії інформації та кодування : навч. посіб. Вінниця : ВНТУ, 2022. 133 с. URL: https://pdf.lib.vntu.edu.ua/books/2023/Majdan_2022_133.pdf (дата звернення: 15.12.2024).
25. Международный стандарт JPEG ISO/IEC 10918.
26. Международный стандарт MPEG ISO CD 11172.
27. Behnam Bani-Egbal. Speeding-Up fractal Image Compression. URL: https://www.uni-konstanz.de/mmsp/fractal2/pdf/Bani94.pdf.lic (дата звернення: 05.06.2025).
28. Fisher Y. Fractal Image Compression. SIGGRAPH 92 Course Notes. e-mail: yfisher@ucsd.edu .
29. Kozhemiako V. P., Maidanuik V. P., Zhukov K. M., Pika S. O. Speeding up of fractal image compression. URL:https://dacemirror.sci-hub.se/proceedings-article/16fc2208630186727b353ab239c535e5/kozhemyako2001.pdf (дата звернення: 05.06.2025).

30. JPEG 2000. Wikipedia. URL: https://en.wikipedia.org/wiki/JPEG_2000 (дата звернення: 05.06.2025).
31. JPEG 2000. Part 1 (Core) jp2 File Format. Sustainability of Digital Formats: Planning for Library of Congress Collections. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000143.shtml (дата звернення: 15.12.2024).
32. JP2 – Image https://docs.fileformat.com/ru/image/jp2/ File Format. URL: (дата звернення: 15.12.2024).
33. JP2 File: The Ultimate Guide to JP2 Files. PaintShop Pro. URL: https://www.paintshoppro.com/en/pages/jp2-file/ (дата звернення: 15.12.2024).
34. JPEG 2000 Signature Format: Documentation & Recovery Example. Active File Recovery. URL: https://www.file-recovery.com/jp2-signature format.htm (дата звернення: 15.12.2024).
35. Ruiz V. G. The JPEG2000 standard (ISO/IEC 15444-1). URL: https://vicente-gonzalez-ruiz.github.io/JPEG2000/ (дата звернення: 15.12.2024).
36. Marcellin M. W., Gormish M. J., Bilgin A., Boliek M. P. An Overviev of JPEG-2000. https://uweb.engr.arizona.edu/~bilgin/publications/DCC2000.pdf . URL: (дата звернення: 15.12.2024).
37. Майданюк В. П., Пєтух А. М. Обробка сигналів : навч. посіб. Вінниця : ВНТУ, 2012. 144 с.
38. Бабак В. П., Хандецький В. С., Шрюфер Е. Обробка сигналів : підручник. К. : Либідь, 1996. 392 с.
39. Furht B., Akar E., Andrews W. A. Digital Image Processing: Practical Approach. ISSN 2191-5768 ISSN 2191-5776 (electronic) SpringerBriefs in Computer Science ISBN 978-3-319-96633-5 ISBN 978-3-319-96634-2 (eBook) https://doi.org/10.1007/978-3-319-96634-2 Library of Congress Control Number: 2018952345 © The Author(s), under exclusive licence to Springer Nature Switzerland AG 2018.
40. Broughton S. A., Bryan K. Discrete Fourier Analysis and Wavelets: Applications to Signal and Image Processing. 2nd Edition. Wiley,2018. 465 p.
41. Top Python libraries for image processing. URL: https://www.geeksforgeeks.org/python/top-python-libraries-for-image-processing/ (дата звернення: 15.01.2025).
42. Документація Python Imaging Library (Pillow). URL: https://pillow.readthedocs.io (дата звернення: 15.01.2025).
43. OpenCV документація. URL: https://docs.opencv.org (дата звернення: 15.01.2025).
44. Best Open Source Image Processing Libraries 2025. URL: https://sourceforge.net/directory/image-processing-libraries/ (дата звернення: 15.01.2025).
45. FFmpeg Codecs Documentation. URL: https://ffmpeg.org/ffmpeg-codecs.html (дата звернення: 15.01.2025).

Електронне навчальне видання

Володимир Павлович Майданюк,
Олександр Никифорович Романюк

Теорія та програмне забезпечення кодування зображень

Навчальний посібник

Рукопис оформив *В. Майданюк*

Редактор *Т. Старічек*

Оригінал-макет виготовила *Т. Старічек*

Підписано до видання 20.01.2026 р.
Гарнітура Times New Roman.
Зам. № Р2026-008
Видавець та виготовлювач
Вінницький національний технічний університет,
Редакційно-видавничий відділ.
ВНТУ, ГНК, к. 114.
Хмельницьке шосе, 95,
м. Вінниця, 21021.
press.vntu.edu.ua;
Email: irvc.vntu@gmail.com
Свідоцтво суб'єкта видавничої справи
серія ДК No 3516 від 01.07.2009 р.