МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

кафедра комп’ютерної інженерії та інформаційних технологій

КВАЛІФІКАЦІЙНА РОБОТА

на тему
Алгоритми дизерингу зображень

Виконав: студент групи 2П-20
Спеціальності
121 – «Інженерія програмного забезпечення»

Володимир ТУРЕНКО

Керівник:
Станіслав МАРЧЕНКО

Черкаси 2024

ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ БІЗНЕС-КОЛЕДЖ

Кафедра комп'ютерної інженерії та інформаційних технологій
(повна назва випускової кафедри)

Спеціальність 121 "Інженерія програмного забезпечення"
(шифр і назва спеціальності)

Освітня програма Інженерія програмного забезпечення
(назва освітньої програми)

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерної інженерії та
інформаційних технологій
(назва кафедри)
_____ Хотунов В.І.
(підпис) (ПІБ)
«_____»_____2023 р.

З А В Д А Н Н Я

НА ВИПУСКНУ РОБОТУ СТУДЕНТУ

Туренку Володимиру Вікторовичу
(прізвище, ім'я, по батькові студента)

1. Тема випускної роботи Алгоритми дизерингу зображень
Керівник роботи Марченко Станіслав Віталійович, спеціаліст І категорії
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від " 13 " жовтня 2023 року № 65У.
2. Строк подання студентом випускної роботи 03.06.2024
3. Вихідні дані до випускної роботи мова програмування Python, середовище розробки Anaconda / Jupyter
4. Зміст випускної роботи (перелік питань, які потрібно розробити) дизеринг зображень: сучасний стан та перспективи (дизеринг у цифровій обробці сигналів, галузі застосування дизерингу, огляд інструментальних засобів для дизерингу зображень), Огляд алгоритмів дизерингу зображень (алгоритми упорядкованого дизерингу, алгоритми випадкового дизерингу, особливості програмної реалізації алгоритмів), порівняльний аналіз алгоритмів дизерингу зображень (методика порівняння алгоритмів, порівняння детермінованих та стохастичних версій алгоритмів, порівняння продуктивності та витрат ресурсів).
5. Дата видачі завдання 15.09.2023р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Терміни виконання етапів	Примітка про виконання
1	Вступ	20.10.2023	
2	Розділ 1. Дизеринг зображень: сучасний стан та перспективи	22.12.2023	
3	Розділ 2. Огляд алгоритмів дизерингу зображень	15.03.2024	
4	Розділ 3. Порівняльний аналіз алгоритмів дизерингу зображень	15.05.2024	
5	Висновки	17.05.2024	
6	Оформлення випускної роботи (чистовий варіант)	27.05.2024	
7	Здача випускної роботи на кафедру для рецензування (за 14 днів до захисту)	31.05.2024	
8	Перевірка випускної роботи на наявність ознак плагіату (за 10 днів до захисту)	03.06.2024	
9	Подання випускної роботи на затвердження завідувачу кафедри (за 7 днів до захисту)	06.06.2024	

Студент _____ Туренко В.В.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Марченко С.В.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Дизеринг – це процес застосування випадкового шуму або патерну, щоб зменшити або приховати артефакти при відтворенні цифрових зображень або звуків, особливо в умовах обмеженого роздільної здатності або кількості кольорів. Актуальність цього процесу може залежати від конкретного контексту або завдання. Дизеринг залишає актуальним у візуальному мистецтві, де створення ефекту "плавного" переходу між кольорами або пікселями може бути важливим для досягнення певної естетичної мети. У друкарській та дизайнерській сферах дизеринг може бути важливим для зменшення артефактів при друку на пристроях з обмеженими можливостями кольорової палітри або роздільної здатності. У відео- та аудіотехнологіях дизеринг може використовуватися для зменшення шуму та артефактів при кодуванні або стисненні медіаданих.

У кваліфікаційній роботі проаналізовано й програмно реалізовано різні алгоритми дизерингу, включаючи детерміновані й випадкові підходи. Виявлено, що кожен метод має свої переваги та недоліки, залежно від контексту використання та вимог до якості зображення. Слід зазначити, що випадкові методи дизерингу показали хороші результати в порівнянні з детермінованими методами, особливо в контексті зменшення візуальних артефактів і збереження природності зображення.

У роботі також було продемонстровано, що використання випадкових методів може бути ефективним для різноманітних типів зображень, включаючи фотографії та різні технічні зображення, які формуються з різним освітленням.

Ключові слова: дизеринг зображення, упорядкований дизеринг, випадковий дизеринг, MSE, PSNR.

ABSTRACT

Dithering is the process of applying random noise or patterning to reduce or hide artifacts in the reproduction of digital images or sounds, especially under conditions of limited resolution or number of colors. The relevance of this process may depend on the specific context or task. Dithering remains relevant in the visual arts, where creating the effect of a "smooth" transition between colors or pixels can be important to achieve a certain aesthetic goal. In the printing and design industries, dithering can be important to reduce artifacts when printing on devices with limited color palette or resolution capabilities. In video and audio technologies, dithering can be used to reduce noise and artifacts when encoding or compressing media data.

In the qualification work, various dithering algorithms, including deterministic and random approaches, were analyzed and software implemented. Each method has been found to have its own advantages and disadvantages, depending on the context of use and image quality requirements. It should be noted that random dithering methods have shown good results compared to deterministic methods, especially in the context of reducing visual artifacts and preserving the naturalness of the image.

The paper also demonstrated that the use of random methods can be effective for a variety of image types, including photographs and various technical images that are formed under different lighting conditions.

Keywords: image dithering, ordered dithering, random dithering, MSE, PSNR.

ЗМІСТ

ВСТУП

Актуальність обраної теми. Дизеринг зображень – це процес створення зображень з низькою роздільною здатністю, які потім масштабуються до високої роздільної здатності. Дизеринг створює ілюзію глибини кольору на зображеннях з обмеженою колірною палітрою. По суті, він зменшує смуги кольорів і відтінків на зображенні.

Основні галузі використання дизерингу зображень націлені на оптимізацію передачі інформації на великі відстані та колірного збагачення для відновлення початкового вигляду подібної інформації. Основними недоліками такого підходу будуть втрата деякої частини графічної інформації, зокрема при стисненні з втратами, а також потреба в додаткових ресурсах для підготовки та відновлення інформації. Дизеринг дозволяє ефективно використовувати обмежену кількість кольорів чи відтінків, проте це компроміс, який може призвести до втрати деякої якості графіки порівняно з використанням повного спектру кольорів.

Незважаючи на класичність технології та методів дизерингу, вони знаходять нові й нові галузі застосування, зокрема, допомагають в роботі лідара, удосконалюючи вимірювання глибини в отриманих зображеннях. Різноманітність алгоритмів дизерингу зображень також ставить питання щодо доцільності та оптимальності застосування вже існуючих чи розробки нових підходів до виконання в конкретних сценаріях використання. Звідси, порівняльний аналіз алгоритмів дизерингу є корисним та актуальним завданням.

Об’єкт дослідження. Об’єктом дослідження виступають алгоритми, спрямовані на реалізацію дизерингу зображень.

Предмет дослідження. Предметом дослідження є методи та підходи для реалізації та порівняльного аналізу алгоритмів дизерингу зображень, оцінювання впливу характеристик вхідних зображень на результати дизерингу.

Мета дослідження. Мета дослідження полягає в порівнянні різних алгоритмів за різними критеріями, таких як якість зображення, розмір файлу, швидкість обробки, простота реалізації та інші.

Завдання дослідження. Для досягнення мети необхідно вирішити такі практичні завдання:

1) виконати огляд сучасних досліджень, можливостей застосування та інструментів дизерингу зображень;
2) виокремити перелік актуальних та перспективних алгоритмів реалізації дизерингу зображень та програмно реалізувати їх;
3) визначити критерії порівняння алгоритмів та результатів їх роботи, проаналізувати застосовність алгоритмів у різних сценаріях і виробити відповідні рекомендації.

РОЗДІЛ 1
ДИЗЕРИНГ ЗОБРАЖЕНЬ: СУЧАСНИЙ СТАН ТА ПЕРСПЕКТИВИ

1.1. Дизеринг у цифровій обробці сигналів

Дизеринг у цифровій обробці сигналів є процесом додавання шуму до сигналу для покращення його якості або для вирішення певних проблем. Основні методи й алгоритми дизерингу можуть використовуватися для зменшення роздільної здатності сигналу з мінімальною втратою якості [1–3]. Це корисно при стисненні аудіо або відео для заощадження простору зберігання даних загалом.

Використання дизерингу сигналів для стабілізації нелінійних систем керування є добре відомим і часто використовуваним методом (рисунок 1.1).

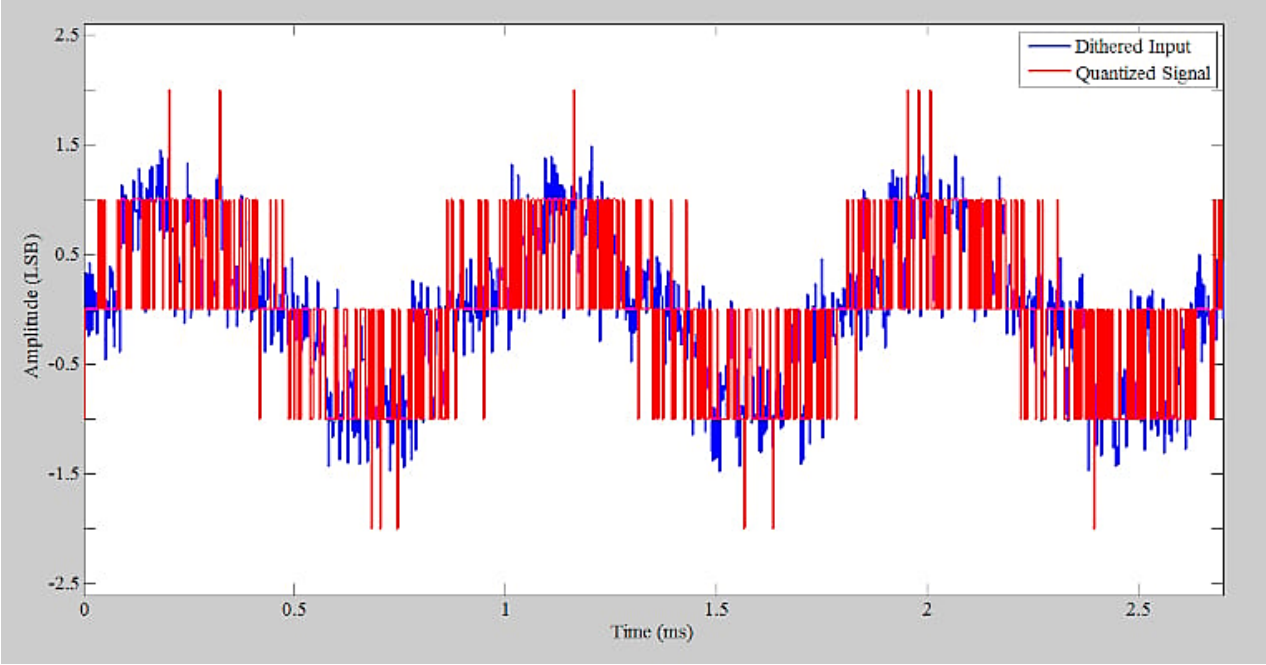


Рисунок 1.1 – Приклад дизерингу сигналу

Ідея полягає в тому, що шляхом введення відповідним чином вибраного високочастотного сигналу в контур керування, нелінійний сектор ефективно звужується, і за допомогою такого підходу можна стабілізувати систему. Теоретичне обґрунтування цієї ідеї для систем з неперервними нелінійностями було отримано Замесом і Шнейдором, і Моссахебом. Відомо, що налаштування дизерингу загальних негладких систем обмежується методами наближеного проєктування, які в основному базуються на описі функцій [4–6]. Також

додавання шуму (дизерингу) може допомогти приховати артефакти, які можуть виникнути в результаті цифрової обробки або стиснення сигналу. Дизеринг надає можливість зробити звук або зображення більш природними та приємними для сприйняття.

Для дизерингу аудіо сигналу, наприклад, можна використовувати випадковий шум низької амплітуди, який додається до сигналу перед квантуванням (рисунок 1.2). Це допомагає зменшити ефект ступінчастості [7–9].

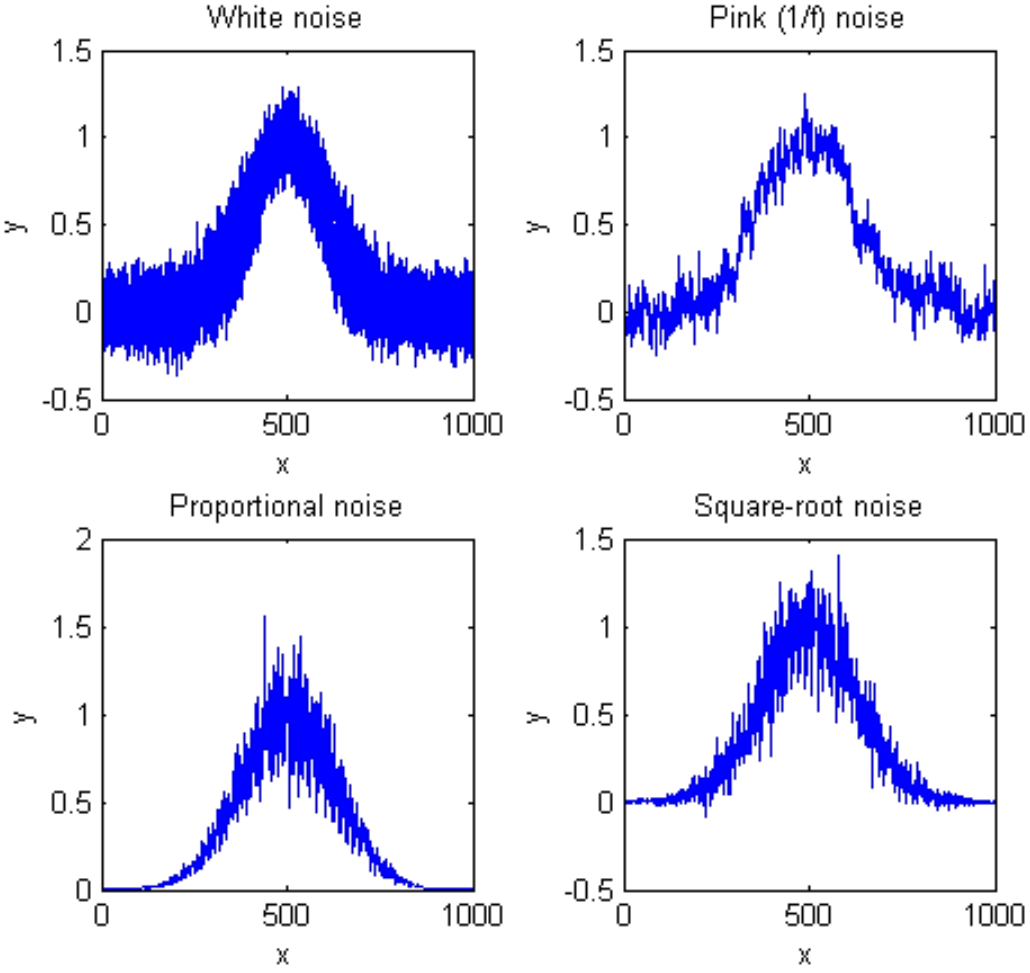


Рисунок 1.2 – Різні види шуму, які спотворюють сигнал

У цифровій обробці сигналів часто застосовуються спеціальні алгоритми дизерингу, такі як алгоритми Флойд-Стейнберга або дифузного дизерингу, які ефективно керують додаванням шуму для досягнення бажаного ефекту, як показано на рисунку 1.3.

Рисунок 1.3 – Застосування алгоритмів поширення похибки
в задачі дизеригну зображень

Слід зазначити, що важливо контролювати рівень шуму, що додається, щоб не перевантажити сигнал непотрібним шумом або навпаки, щоб шум був недостатньо помітний [10–12].

Після дизерингу сигнал / зображення може бути проаналізований та оцінений на предмет покращення якості або досягнення цілей обробки.

1.2. Галузі застосування дизерингу

Квантування й дирезинг широко використовуються у відеокодуванні або стисненні відео [13–15]. Квантування – це нелінійний процес, який дозволяє

зменшити кількість бітів, необхідних для кодування певної інформації. Загалом, квантування є процесом із втратами. Існує багато методів для зменшення помилок квантування в квантованому сигналі. Для стиснення відео помилки квантування можуть мати багато негативних наслідків для сприйняття основного відео [16; 17]. Деякі з ефектів можуть бути відразу помітні користувачеві, тоді як інші можуть бути помітні не відразу, але, все ж, важливі для сприйняття якості відео. Для стиснення відео на основі блоків (тих, що використовуються у стандартах стиснення відео, таких як MPEG-2, MPEG-4, H.264 або H.262), найбільш помітними ефектами квантування можуть бути смуги та блокові артефакти, особливо при стисненні відео з низьким бітрейтом [18–20]. Артефакти блоків можуть бути безпосередньо спричинені схемами стиснення відео на основі блоків, де для кодування відео використовуються блоки пікселів. Якщо квантування надто сильне, локально може бути видимим лише рівномірний одноколірний блок і, таким чином, втрачаються всі деталі вхідного кадру зображення. Артефакти смуг можуть бути схожі на артефакти блоків, але смуги можуть бути видимими на всіх бітрейтах, включаючи навіть високі. Смуги можуть бути особливо помітними в областях із плавним градієнтом кольору, наприклад, чисте небо у фільмі, або пейзажі у створеній комп’ютером графіці фільму, наприклад, у мультфільмі. Помилки квантування можуть виникати в ряді моментів під час процесу кодування відео.

Зменшена кількість бітів може викликати помилки квантування безпосередньо в нестиснутому джерелі на вході ланцюжка кодування. За певних ситуацій постачальник вихідного відео може вжити відповідних заходів для покриття артефактів квантування та/або смуг на вихідному відео чи кадрах зображень. Однак в інших ситуаціях постачальник може цього не зробити. Коли доступна лише версія відео з низькими бітрейтами на канал, а не вихідне відео високої якості, процесор може виконати «сліпий» пошук можливих місць смуг або артефактів блокування. Артефакти смуг навіть у високоякісних схемах кодування , може бути чітко видимим для користувача. Це особливо вірно для кодування сцен фільму великих областей з рівномірним повільним градієнтом

кольору, наприклад, блакитне небо або вицвітання в назвах фільмів. Цей ефект смуги може бути пов’язаний з обмеженнями в моделях блокового кодування в існуючих стандартах, таких як MPEG-2 або H.264. Наявність градієнта кольору, нахил якого є меншим за мінімальний дозволений крок квантування, може, швидше за все, спричинити видимі артефакти на остаточних кадрах зображення. Для маскування смугових артефактів, які можуть бути спричинені квантуванням, можна використовувати техніку змішування. «Змішування» зазвичай розуміється як навмисне застосування шуму до відео чи аудіо, що використовується для рандомізації помилок квантування, щоб запобігти появі великомасштабних шаблонів на зображенні. Крім того, дизеринг зазвичай застосовується глобально до всіх пікселів у відеокадрі перед процесом квантування або повторного квантування, щоб запобігти нелінійним спотворенням. Також кількість шуму, доданого під час згладжування, може залежати від багатьох факторів. Наприклад, чим менша бітова глибина кожного пікселя, тим більше може знадобитися згладжування. Глобальне згладжування, наприклад, додавання однакової кількості шуму до кожного пікселя на основі глобального вимірювання помилки квантування.

Техніка дизерінгу особливо корисна в системах зв’язку. У багатьох комунікаційних програмах вхідним сигналом може бути слабкий сигнал, значно нижчий за повну шкалу аналого-цифрового пристрою (АЦП). Цей малий сигнал виконує відносно невелику кількість кодів АЦП. Якщо ці коди виявляють великі середньоквадратичні помилки (MSE), вихід міститиме значні гармонійні спотворення. Можна звернути увагу, що для широких сигналів помилка DNL певною мірою усереднена. Причина в тому, що широкий сигнал виконує всі коди АЦП. Як наслідок, АЦП, який демонструє повномасштабний SFDR 88 дБFS, може забезпечити лише 80 дБFS SFDR, коли амплітуда сигналу зменшується на 20 дБ нижче повномасштабного значення. У таких випадках техніка змішування може допомогти нам підтримувати продуктивність SFDR АЦП на низьких рівнях сигналу. Слід зазначити, що оскільки вхідний рівень невеликий, можна додати до входу сигнал з дизерингом, не перевантажуючи жоден АЦП.

Обмеження техніки дизерингу. Належний рівень дизерингу, який забезпечує значне покращення SFDR АЦП, залежить від архітектури та інших властивостей цього конкретного АЦП. Поліпшення SFDR також залежить від амплітуди вхідного сигналу, а також амплітуди дизерингу. Слід також зазначити, що за межами певного рівня шуму SFDR може не покращитися суттєво. Як приклад, розглянемо AD6645 від Analog Devices. У цьому пристрої використовується багатоступенева архітектура. У цьому типі архітектури АЦП помилка DNL має повторюваний шаблон, і на графіку DNL виникають деякі стрибки, оскільки вхідний сигнал переходить за межі вхідного діапазону АЦП. На рисунку 1.4 нижче показано графік DNL AD6645 у невеликій частині його вхідного діапазону.

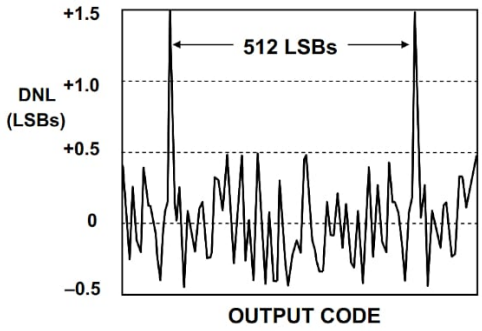


Рисунок 1.4 – Діаграма DNL AD6645 у невеликій частині його вхідного діапазону

1.3. Огляд інструментальних засобів для дизерингу зображень

1.3.1. Вебресурс дизерингу зображень ditherit.com

Даний вебресурс призначений для створення ефекту псевдотональності або зменшення видимості артефактів при зниженні роздільної здатності зображення. Він може бути корисним для тих користувачів, які хочуть експериментувати з дизерингом, створювати зображення із псевдотональним ефектом або бажають зменшити розмір зображення з мінімальною втратою якості початкового зображення.

При роботі з сайтом для початку користувач завантажує зображення на сайт. Головний інтерфейс ресурсу має такий вигляд, який показано на рисунку 1.5.

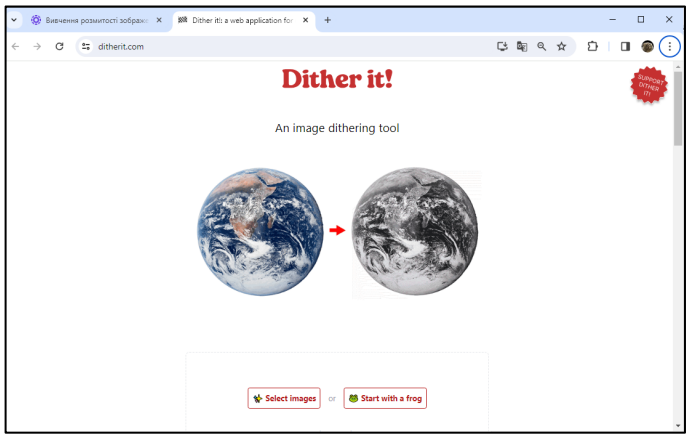


Рисунок 1.5 – Інтерфейс стартової сторінки вебресурсу ditherit.com

Після завантаження зображення користувач ресурсу може обрати різні параметри дизерингу, такі як розмір зображення, тип дизерингу (наприклад, Флойд-Стейнберг, Байєр тощо), рівень дизерингу та інші параметри, як показано на рисунку 1.6. Цей вебресурс застосовує вибрані параметри зображення, додаючи ефект дизерингу. Після обробки зображення вебресурс надає можливість завантажити результат у вигляді нового зображення із застосованим ефектом дизерингу.

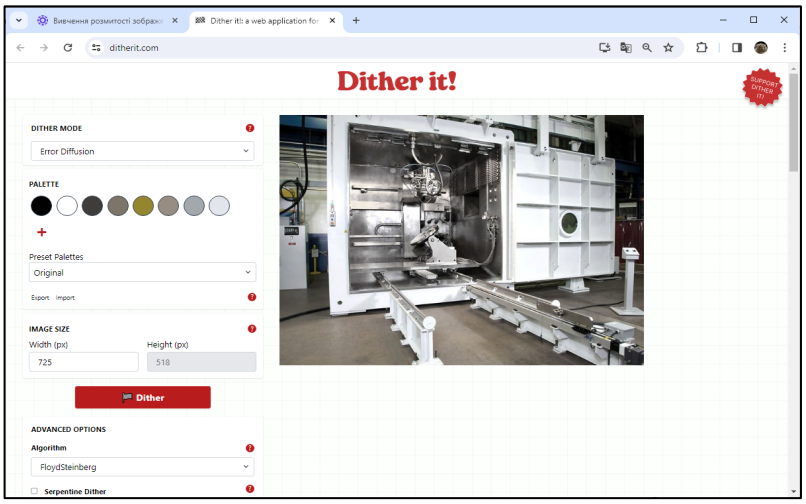


Рисунок 1.6 – Сторінка вибору параметрів застосування дизерингу завантаженого зображення до ресурсу ditherit.com

На рисунку 1.7 показані основні значення параметрів, які можна обрати для створення дизерингу будь-якого обраного користувачем ресурсу зображення.

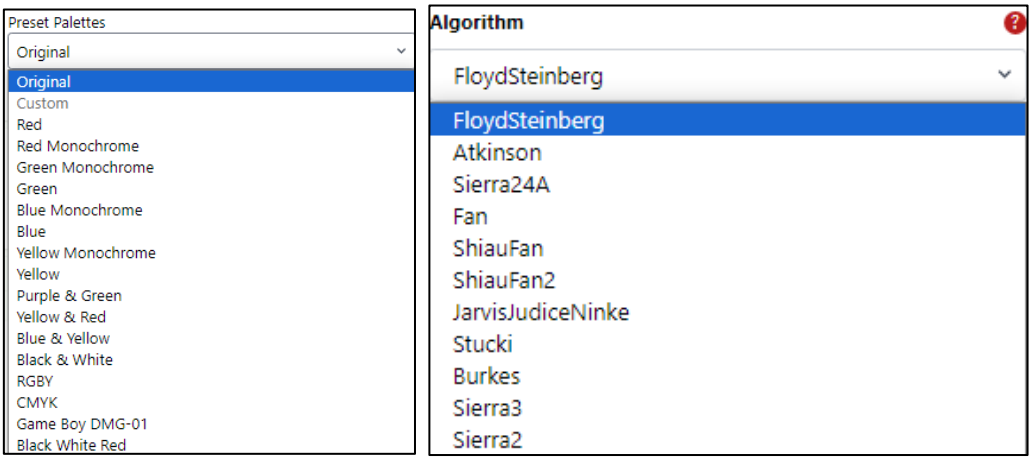


Рисунок 1.7 – Вибір параметру, який полягає у обранні палітри й відомих алгоритмів дизерингу зображення

Слід також зазначити, що палітру кольорів можна обирати в уже сформованому за замовчуванням списку, а можна й з використанням коду, як показано в зразку:

```
[
      {"hex":"#010101"},
      {"hex":"#ffffff"},
      {"hex":"#3e3d3b"},
      {"hex":"#7d756a"},
      {"hex":"#93862f"},
      {"hex":"#978d83"},
      {"hex":"#a3a8ac"},
      {"hex":"#e2e5ec"}
]
```

У наведеному вище коді спочатку вказується система, яка формує палітру «hex», а потім відповідне значення кольору у цій палітрі у 16-й систем числення. Колір має починатись з гештегу. Після гештегу вказується 6 символів з алфавіту 16-ї системи числення. Перші два символи відповідають за силу червоного кольору, другі два символи – за зелену компоненту, останні два символи – за компоненту blue.

Також є можливість здійснювати імпорт й експорт елементів (кольорів) палітри. На рисунку 1.8 присутній фактор зернистості зображення, що є результатом дизеригну за алгоритмом Флойда-Стейнберга.

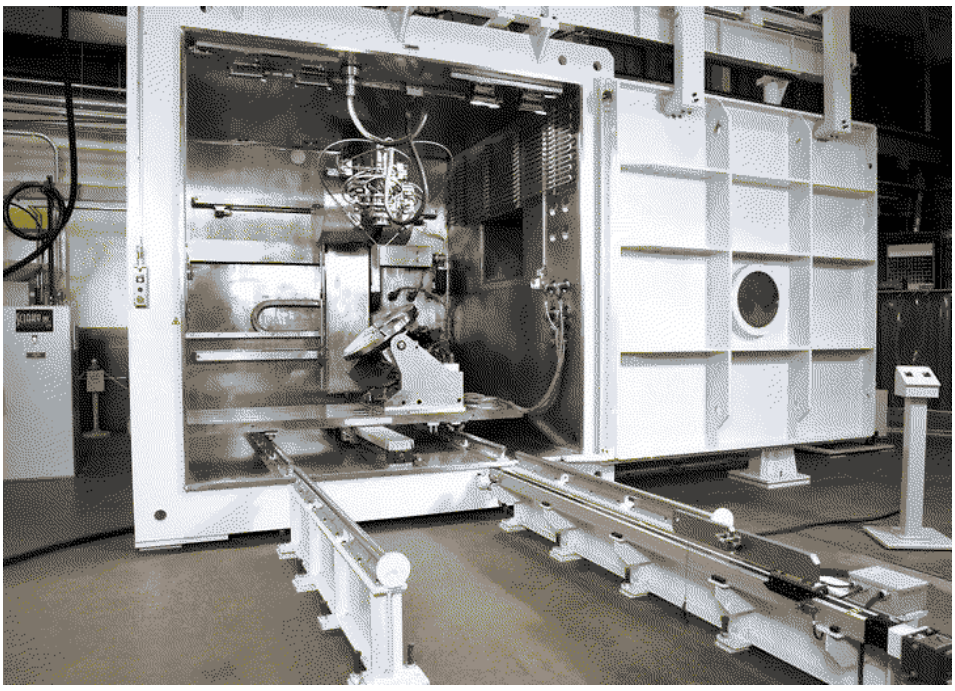


Рисунок 1.8 – Результат дизерингу після застосування алгоритму Флойда-Стейнберга

На рисунку 1.9 показаний результат дизерингу з використанням алгоритму Байєра

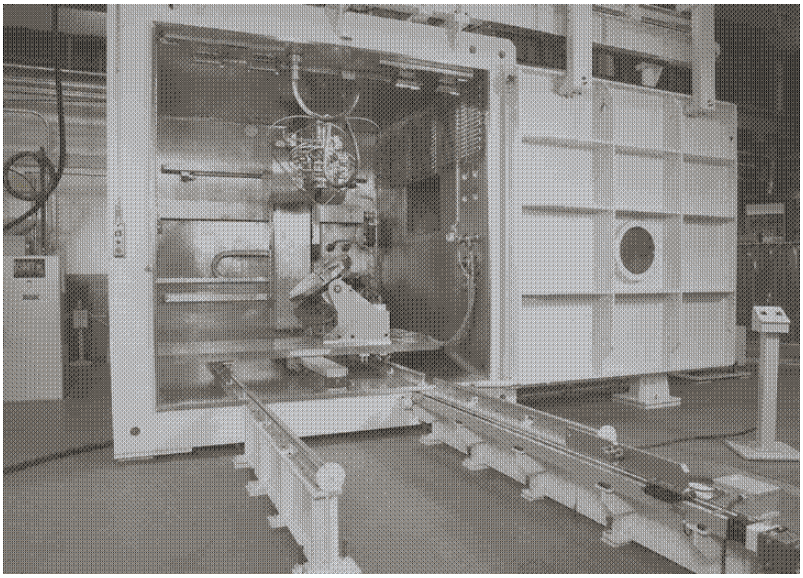


Рисунок 1.9 – Результат дизерингу після застосування алгоритму Байєера

Якщо порівняти рисунок 1.8 і рисунок 1.9, результат на останньому рисунку має більші спотворення відносно оригіналу. Для палітри з двох

кольорів (чорний і білий кольори) відповідні результати показані на рисунку 1.10 і рисунку 1.11.

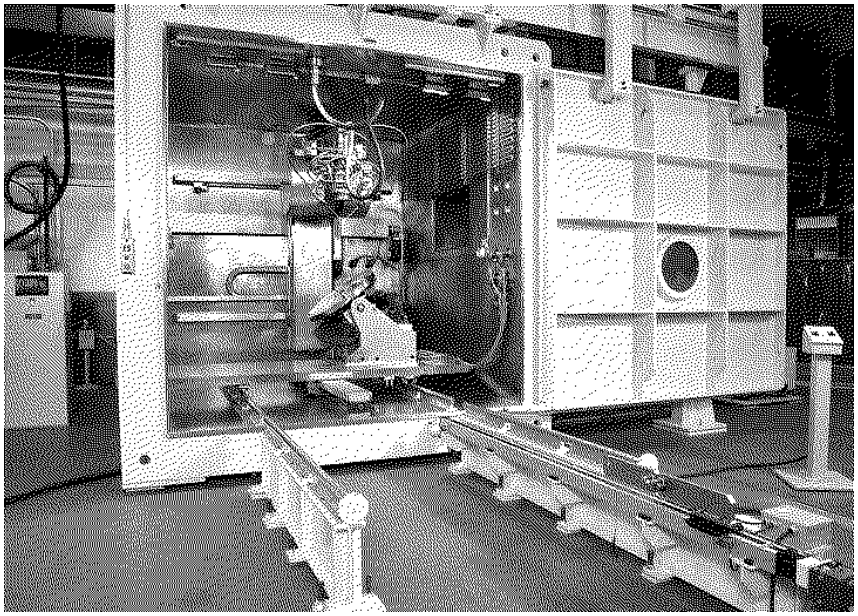


Рисунок 1.10 – Результат дизерингу після застосування алгоритму Флойда-Стейнберга для палітри, яка складається з чорного і білого кольорів

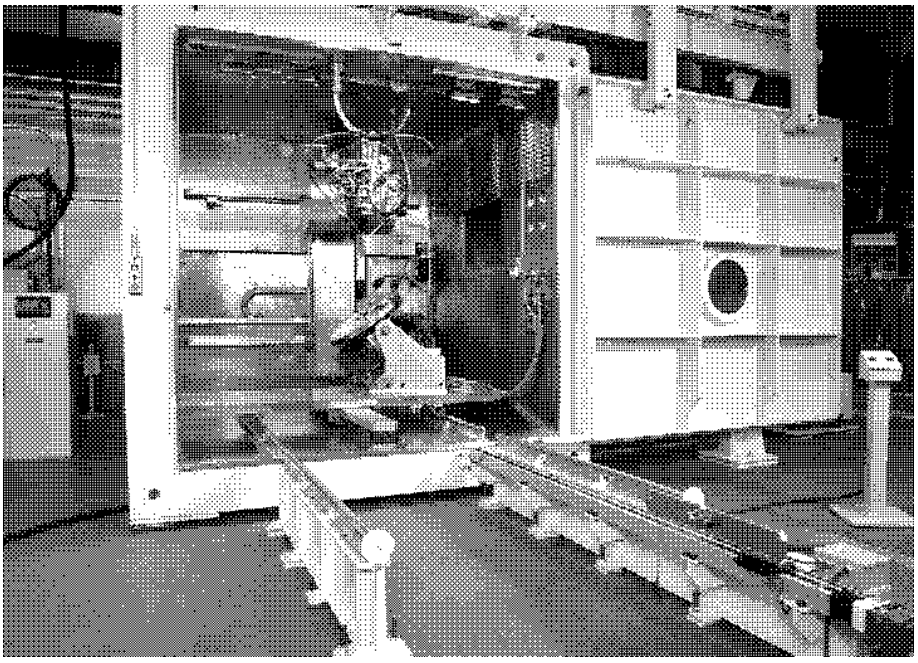


Рисунок 1.11 – Результат дизерингу після застосування алгоритму Байєра для палітри, яка складається з чорного і білого кольорів

Варто також зазначити, що зображення можна масштабувати, як показано на рисунку 1.12.

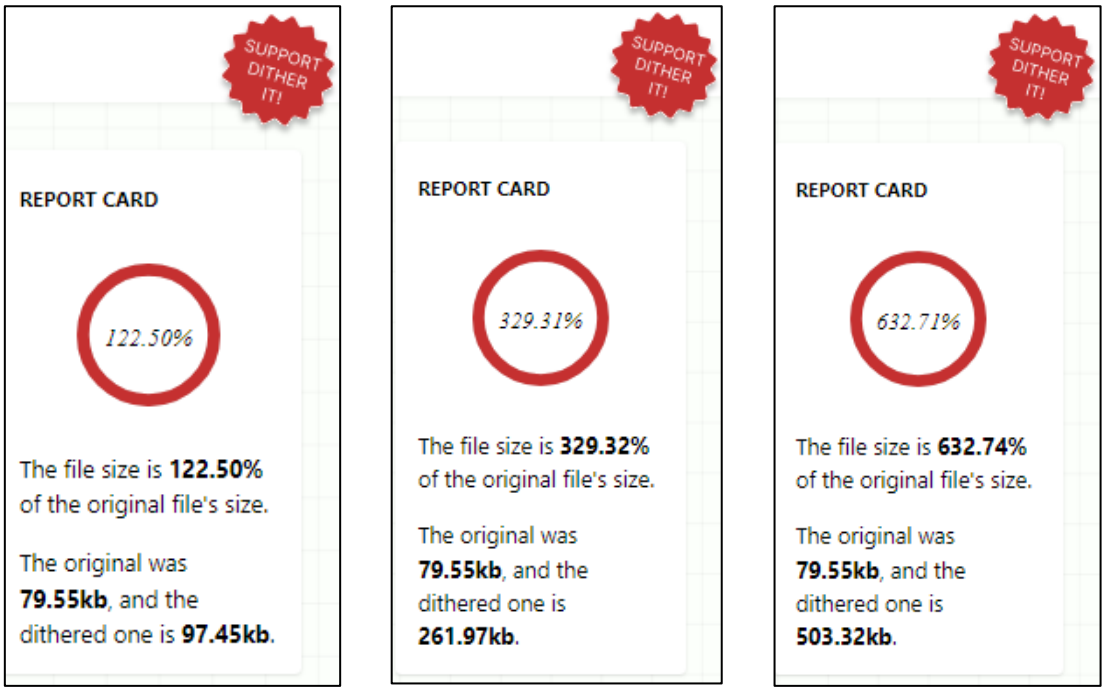


Рисунок 1.12 – Різні масштабування результуючого зображення на основі IMAGE SIZE: Width (px), Height (px)

1.3.2. Графічний редактор Adobe PhotoShop

Adobe PhotoShop – це один з найпопулярніших інструментів для редагування зображень та графічного дизайну. Він має широкий спектр функцій і можливостей, які дозволяють користувачам створювати і редагувати графіку з високою якістю.

Складні зображення. Якщо взяти складне зображення, таке як друкована фотографію, то можна побачити кольорові крапки всюди, де є затінення. У процесі друку використовуються кольори з чорнила або тонера принтера для створення проміжних відтінків шляхом згладжування. Офсетний друк використовує фіксовану сітку точок і змінює розмір точки для отримання потрібного відтінку. PhotoShop також може використовувати опцію «дифузійне згладжування» для генерації випадкового шаблону точок, які мають однаковий розмір, але розташовані ближче або далі одна від одної, щоб отримати відтінок, який найбільше відповідає оригінальному зображенню. Ці два види згладжування створюють складні зображення високої якості залежно від кількості точок на дюйм.

Прості зображення. Прості зображення, як-от логотипи або шрифти, містять невелику кількість кольорів із гладкими поверхнями й гострими краями. Вони відображаються краще, коли PhotoShop використовує менше кольорів, оскільки тоді кольорові області є плоскими, як вони були задумані. Формат GIF – це стиснутий графічний формат без втрат, який підтримує лише 256 кольорів. Під час збереження у форматі GIF згладжування PhotoShop намагається імітувати затінення, завдяки чому плоскі кольорові області виглядають неоднозначними. Натомість PhotoShop має зберегти GIF-зображення, використовуючи найближчий із 256 кольорів без будь-якого згладжування.

Чорно-білі зображення. Коли користувач використовує PhotoShop, щоб змінити зображення на чорно-біле, згладжування обмежується двома кольорами чорного та білого по всьому зображенню. Щоб отримати плавне затінення сірого, програма розміщує чорні пікселі поруч із білими. PhotoShop дозволяє вибрати дифузію, згладжування візерунків або шуму чи відсутність змішування. Відсутність згладжування призводить до плоских суміжних ділянок чорного, білого або обмеженої кількості сірого. Змішування шаблону розміщує чорно-білі пікселі в сітці. Дифузійне згладжування призводить до випадкових, але рівномірно розташованих пікселів, а пікселі змішування шуму розташовані нерівномірно. Методи згладжування дають різні ефекти затінення.

Творчий дизеринг. Фотографи іноді використовують згладжування PhotoShop для художніх ефектів. За низької роздільної здатності згладжування надає точкову текстуру зображення. Також можливі різні цікаві візерунки. Якщо додати шум до чорного фону, розмити, перетворити зображення на чорно-біле, а потім розсіяти результат, отримується пунктирний візерунок, який змінюється на ваших очах у вигляді оптичної ілюзії. Різне згладжування призведе до різних візерунків. Для цього існують різні способи.

Спочатку дизеринг робили «вручну». Користувач просто вибирає обмежену палітру кольорів, а потім вручну розміщує кожен піксель. Це вид мистецтва, який вимагає багато практики. Швидшим способом буде взяти ваше вихідне зображення RGB і змінити колірний режим на індексований

колір (Image/Mode/Indexed Color). Це дає змогу зменшити кількість різних кольорів на зображенні. Подивимось на такий плавний градієнт, який показано на рисунку 1.13:

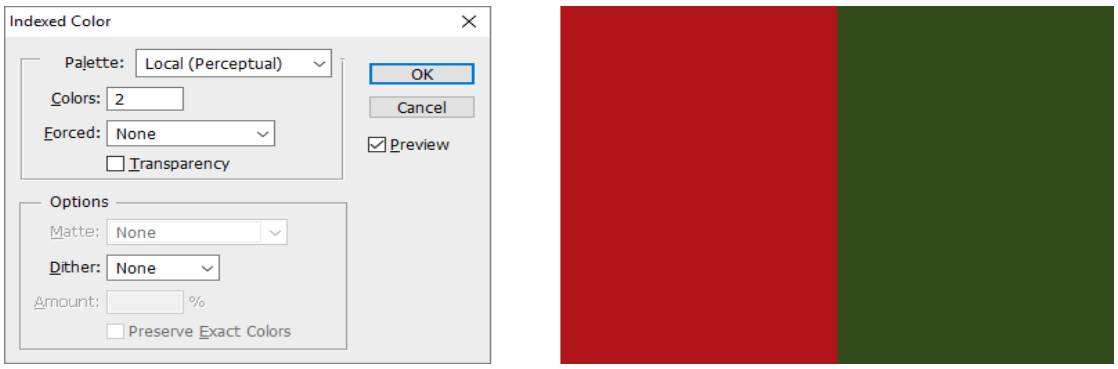
Рисунок 1.13 – Плавний градієнт

Нижче показано, як виглядає перетворення цього градієнта у два кольори за допомогою 4 різних стилів згладжування PhotoShop (рисунок 1.14). Можна далі проекспериментувати з налаштуваннями. У такому випадку, можливо, користувач захоче обрати 256 кольорів, щоб досягти правильного «ретро» відчуття.

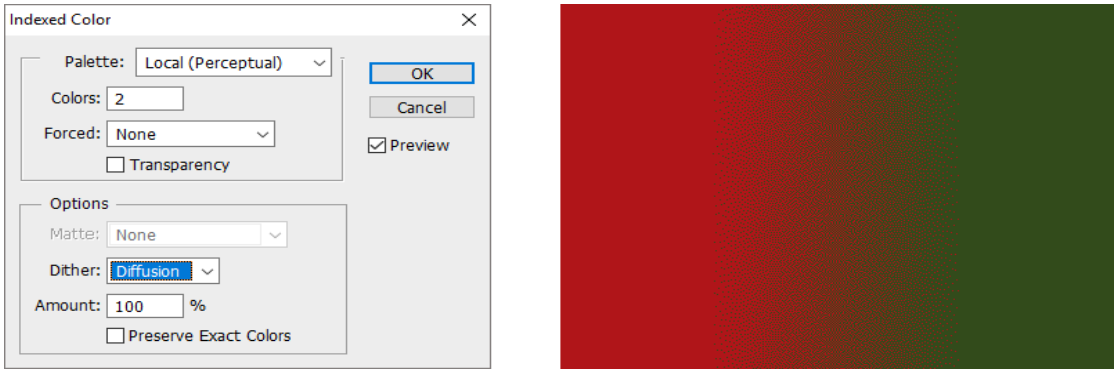
1.3.3. Графічний редактор Gimp

GIMP – це безкоштовний та відкритий редактор зображень з широким вибором прикладних функцій. Цей редактор доступний для операційних систем сімейства Windows, MacOS та Linux та надає багато інструментів для обробки та редагування зображень загалом. GIMP часто використовується як альтернатива платним середовищам редагування зображень, таким як Adobe PhotoShop. GIMP підходить як для початківців, так і для досвідчених користувачів, завдяки своїй гнучкості та потужним інструментам. Нижче наведений короткий опис для створення дизерингу вхідного зображення.

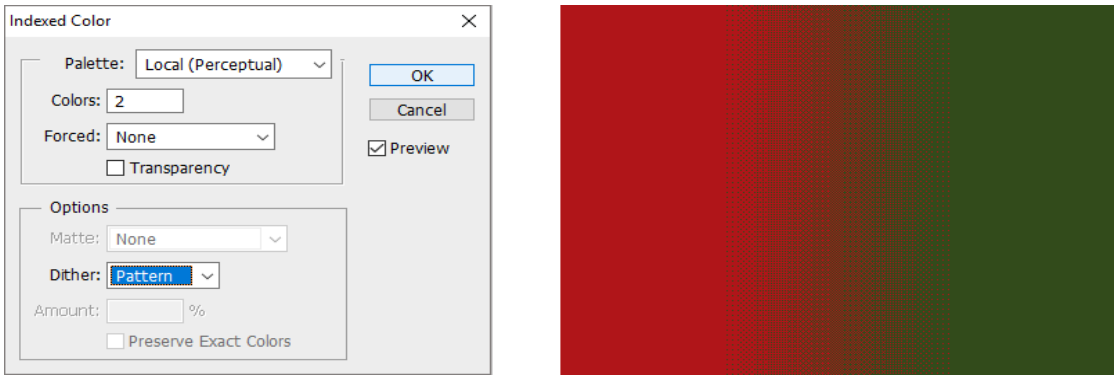
Крок 1. Щоб приступити до процесу дизерингу зображення, користувачу даним програмним забезпеченням потрібно обрати фотографію, цифрову ілюстрацію чи графічну ілюстрацію.



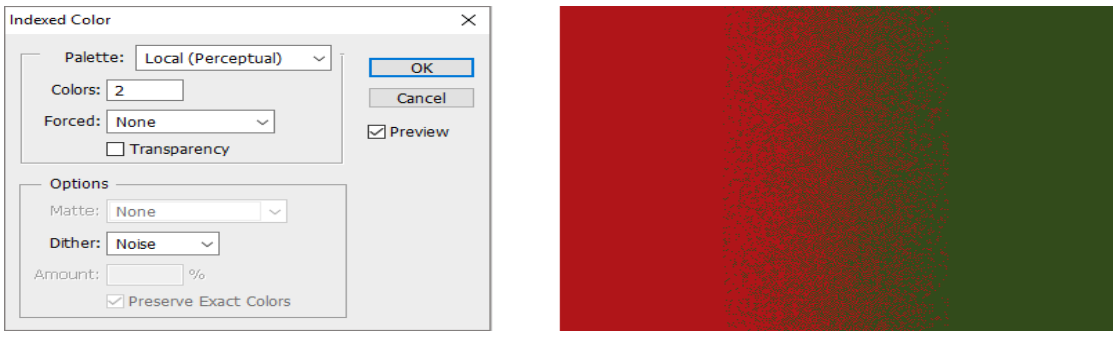
(а)



(б)



(в)



(г)

Рисунок 1.14 – 4 різні стилі згладжування PhotoShop: а) Немає (None); б) Дифузія (Diffusion); в) Візерунок (Pattern); г) Шум (Noise)

Крок 2. На цьому кроці користувачем може бути виконана зміна діапазону кольорів (рисунок 1.15):

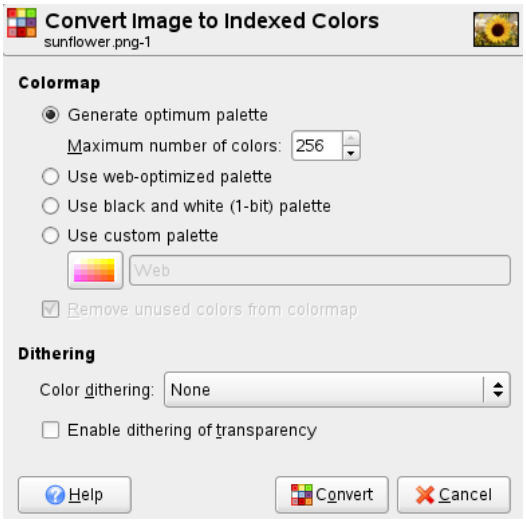


Рисунок 1.15 – Вибір зміни кольорів (кольорова палітра)

Тут обирається і застосовується конкретний колірний режим до вибраної фотографії. Користувач може обрати стандартні режими, такі як RGB, CMYK або відтінки сірого, за умови, що в обраному зображенні можна мінімізувати типи кольорів. Для цього користуються опціями зменшення кольорів у програмному забезпеченні – «Зображення → Режим → Індекс». В останньому меню слід обрати «Створити оптимальну палітру» та встановити потрібну кількість кольорів. Далі можна використовувати такі інструменти вибору GIMP, як Free Select, щоб застосувати вибраний колір до вибраних областей або загального зображення. Слід використовувати опцію попереднього перегляду, щоб спостерігати за загальним впливом маніпуляції кольором на ваших зображеннях. Варто переконатись у тому, що він узгоджується з ефектом згладжування, який потрібно застосувати в наступних кроках. Можна проекспериментувати з різними параметрами та алгоритмами розмитості, доступними у вашому програмному забезпеченні, щоб оцінити шалені припущення щодо результатів, які матиме ваше зображення.

Крок 3. Вибір інструменту для ефекту дизерингу (рисунок 1.16)

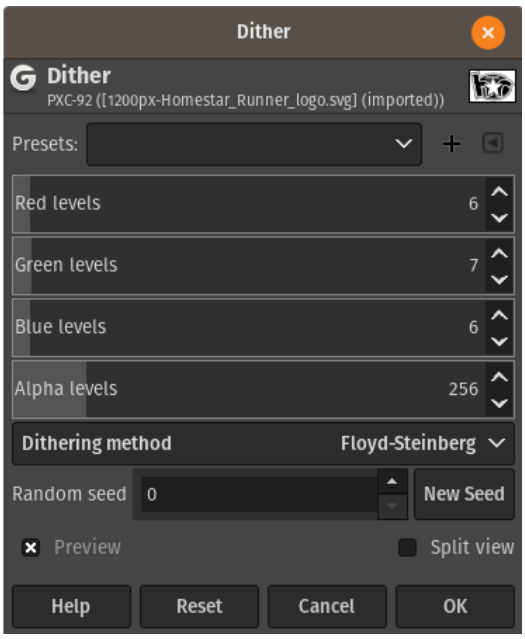


Рисунок 1.16 – Вибір інструменту для ефекту дизерингу

Отже, за результатами огляду концепції дизерингу та сучасних засобів її впровадження, було виявлено значне поширення технології в різних галузях та широке різноманіття способів реалізації. Звідси, окремий інтерес становить порівняльний аналіз та цільові рекомендації стосовно застосування алгоритмів дизерингу.

РОЗДІЛ 2
ОГЛЯД АЛГОРИТМІВ ДИЗЕРИНГУ ЗОБРАЖЕНЬ

2.1. Алгоритми упорядкованого дизерингу

2.1.1. Алгоритм розсіювання помилок Флойда-Стейнберга

Обробка зображень розглядається як джерело масивного паралелізму даних, придатного для обробки таким пристроєм, як GPU [1; 5]. Звичайні операції фільтрації зображень забезпечують високий паралелізм, оскільки кожен вхідний піксель створює одне вихідне піксельне значення незалежно або шляхом обробки невеликої області навколо себе у вхідному зображенні. Той факт, що вихідні пікселі залежать лише від вхідних пікселів, і що останні не змінюються в результаті обробки, дає змогу обробляти всі пікселі паралельно. Такий паралелізм недоступний, якщо операція має причинно-наслідкову залежність від результату попередніх пікселів [4]. Звідси, вихідне значення «пізніших» вихідних пікселів залежить від суміші значень вхідних пікселів і значень вихідних пікселів, обчислених «раніше». Порядок причинної обробки для 2D-зображення можна визначити, вибравши один його кут як перший піксель, а протилежний кут — як останній у порядку обробки. Потім обробка може тривати вздовж рядків або вздовж стовпців. Етап обробки кожного пікселя може бути лінійною операцією або включати нелінійні обчислення [8].

Основний підхід в алгоритмі Флойда-Стейнберга відбувається так. Для кожного пікселя накопичується помилка від його сусідів, причому ця помилка є зваженою, матриця ваг подається у вигляді:

$$G = \frac{1}{16}\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0(*) & 7 \\ 3 & 5 & 1 \end{pmatrix}. \qquad (2.1)$$

Матриця (2.1) накладається на центральний піксель, який позначено знаком зірочки «*». Для початку роботи алгоритму задається палітра кольорів, на основі якої буде формуватися нове зображення. У такому випадку для поточного пікселя, який обробляється знаходиться найближчий колір $NewColor$ з тієї палітри, на якій буде будуватись вихідне зображення:

$$err = s_{ij} - NewColor, \tag{2.2}$$

Помилка у (2.2) поширюється на сусідні пікселі на основі (2.1) за таким принципом. Візьмемо фрагмент зображення, що записується у вигляді:

$$I = \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix}. \tag{2.3}$$

Слід зазначити, що у (2.3) значення s_{ij} $i, j \in \overline{1;3}$ можуть приймати цілі значення від 0 до 255 або значення від 0 до 1. На основі даних двох матриць знаходиться поелементний добуток, результат якого є оновлені значення інтенсивності пікселів за даним алгоритмом. Тобто маємо такий вираз:

$$\begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix} = \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix} + err * G =$$
$$= \begin{pmatrix} s_{11} & s_{12} & s_{13} \\ s_{21} & s_{22} & s_{23} \\ s_{31} & s_{32} & s_{33} \end{pmatrix} + \frac{err}{16} \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 7 \\ 3 & 5 & 1 \end{pmatrix} =$$
$$= \begin{pmatrix} s_{11} + \frac{err}{16} \cdot 0 & s_{12} + \frac{err}{16} \cdot 0 & s_{13} + \frac{err}{16} \cdot 0 \\ s_{21} + \frac{err}{16} \cdot 0 & s_{22} + \frac{err}{16} \cdot 0 & s_{23} + \frac{err}{16} \cdot 77 \\ s_{31} + \frac{err}{16} \cdot 3 & s_{32} + \frac{err}{16} \cdot 5 & s_{33} + \frac{err}{16} \cdot 1 \end{pmatrix}. \tag{2.4}$$

Отриманий результат (2.4) можна записувати не лише в матричному вигляді, тим самим зекономити трохи пам’яті на зберіганні непотрібних даних, якими є нульові елементи матриці.

2.1.2. Алгоритм Джарвіса–Джудіса–Нінке

Основний підхід в алгоритмі Джарвіса–Джудіса–Нінке передбачає, що для кожного пікселя накопичується помилка від його сусідів, причому дана помилка є зваженою, матриця ваг подається у вигляді:

$$G = \frac{1}{48} \begin{pmatrix} 0 & 0 & 0(*) & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{pmatrix}. \tag{2.5}$$

Матриця (2.5) накладається на центральний піксель, який позначено знаком зірочки «*». Для початку роботи алгоритму задається палітра кольорів, на основі

якої буде формуватися нове зображення. У такому випадку для поточного пікселя, який обробляється, знаходиться найближчий колір $NewColor$ з тієї палітри, на якій буде будуватись вихідне зображення:

$$err = s_{ij} - NewColor, \tag{2.6}$$

Помилка у (2.6) поширюється на сусідні пікселі на основі (2.1) за таким принципом:

Візьмемо фрагмент зображення, що записується у вигляді:

$$I = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix}. \tag{2.7}$$

Варто зазначити, що у (2.7) значення s_{ij} $i \in \overline{1;3}, j \in \overline{1;5}$ можуть приймати цілі значення від 0 до 255 або значення від 0 до 1. На основі даних двох матриць знаходиться поелементний добуток, результат якого є оновлені значення інтенсивності пікселів за даним алгоритмом. Тобто маємо такий вираз:

$$\begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} + err * G =$$

$$= \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} + \frac{err}{48} \begin{pmatrix} 0 & 0 & 0(*) & 7 & 5 \\ 3 & 5 & 7 & 5 & 3 \\ 1 & 3 & 5 & 3 & 1 \end{pmatrix} = \tag{2.8}$$

$$= \begin{pmatrix} s_{11} + \frac{err}{48} \cdot 0 & s_{12} + \frac{err}{48} \cdot 0 & s_{13} + \frac{err}{48} \cdot 0 & s_{14} + \frac{err}{48} \cdot 7 & s_{15} + \frac{err}{48} \cdot 5 \\ s_{21} + \frac{err}{48} \cdot 3 & s_{22} + \frac{err}{48} \cdot 5 & s_{23} + \frac{err}{48} \cdot 7 & s_{24} + \frac{err}{48} \cdot 5 & s_{25} + \frac{err}{48} \cdot 3 \\ s_{31} + \frac{err}{48} \cdot 1 & s_{32} + \frac{err}{48} \cdot 3 & s_{33} + \frac{err}{48} \cdot 5 & s_{34} + \frac{err}{48} \cdot 3 & s_{35} + \frac{err}{48} \cdot 1 \end{pmatrix}.$$

Аналогічно, отриманий результат (2.8) можна записувати не лише в матричному вигляді, тим самим зекономити трохи пам’яті на зберіганні непотрібних даних.

2.1.3. Алгоритм Пітера Штуки

Основний підхід в алгоритмі Пітера Штуки відбувається так. Для кожного пікселя накопичується помилка від його сусідів, причому дана помилка є зваженою, матриця ваг подається у вигляді:

$$G = \frac{1}{42}\begin{pmatrix} 0 & 0 & 0(*) & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{pmatrix}. \quad (2.9)$$

Матриця (2.9) накладається на центральний піксель, який позначено знаком зірочки «*». Для початку роботи алгоритму задається палітра кольорів, на основі якої буде формуватися нове зображення. У такому випадку для поточного пікселя, який обробляється знаходиться найближчий колір $NewColor$ з тієї палітри, на якій буде будуватись вихідне зображення:

$$err = s_{ij} - NewColor, \quad (2.10)$$

Помилка у (2.10) поширюється на сусідні пікселі на основі (2.1) за таким принципом. Візьмемо фрагмент зображення, що записується у вигляді:

$$I = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix}. \quad (2.11)$$

Слід зазначити, що у (2.11) значення s_{ij} s_{ij} $i \in \overline{1;3}, j \in \overline{1;5}$ можуть приймати цілі значення від 0 до 255 або значення від 0 до 1. На основі даних двох матриць знаходиться поелементний добуток, результатом якого є оновлені значення інтенсивності пікселів за даним алгоритмом. Тобто маємо такий вираз:

$$\begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} + err * G =$$
$$= \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} + \frac{err}{42}\begin{pmatrix} 0 & 0 & 0(*) & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \\ 1 & 2 & 4 & 2 & 1 \end{pmatrix} = \quad (2.12)$$
$$= \begin{pmatrix} s_{11} + \frac{err}{42} \cdot 0 & s_{12} + \frac{err}{42} \cdot 0 & s_{13} + \frac{err}{42} \cdot 0 & s_{14} + \frac{err}{42} \cdot 8 & s_{15} + \frac{err}{42} \cdot 4 \\ s_{21} + \frac{err}{42} \cdot 2 & s_{22} + \frac{err}{42} \cdot 4 & s_{23} + \frac{err}{42} \cdot 8 & s_{24} + \frac{err}{42} \cdot 4 & s_{25} + \frac{err}{42} \cdot 2 \\ s_{31} + \frac{err}{42} \cdot 1 & s_{32} + \frac{err}{42} \cdot 2 & s_{33} + \frac{err}{42} \cdot 4 & s_{34} + \frac{err}{42} \cdot 2 & s_{35} + \frac{err}{42} \cdot 1 \end{pmatrix}.$$

Отриманий результат (2.12), як і вище, можна записувати не лише в матричному вигляді, тим самим зекономити трохи пам’яті на зберіганні непотрібних даних.

2.1.4. Алгоритм Аткінсона

Основний підхід в алгоритмі Аткінсона передбачає, що для кожного пікселя накопичується помилка від його сусідів, причому дана помилка є зваженою, матриця ваг подається у вигляді:

$$G = \frac{1}{8}\begin{pmatrix} 0 & 0(*) & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}. \tag{2.13}$$

Матриця (2.13) накладається на центральний піксель, який позначено знаком зірочки «*». Для початку роботи алгоритму задається палітра кольорів, на основі якої буде формуватися нове зображення. У такому випадку для поточного пікселя, який обробляється знаходиться найближчий колір $NewColor$ з тієї палітри, на якій буде будуватись вихідне зображення:

$$err = s_{ij} - NewColor, \tag{2.14}$$

Помилка у (2.14) поширюється на сусідні пікселі на основі (2.1) за таким принципом. Візьмемо фрагмент зображення, що записується у вигляді:

$$I = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \end{pmatrix}. \tag{2.15}$$

Варто зазначити, що у (2.15) значення s_{ij} s_{ij} $i \in \overline{1;3}, j \in \overline{1;4}$ можуть приймати цілі значення від 0 до 255 або значення від 0 до 1. На основі даних двох матриць знаходиться поелементний добуток, результат якого є оновлені значення інтенсивності пікселів за даним алгоритмом. Тобто маємо такий вираз:

$$\begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \end{pmatrix} = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \end{pmatrix} + err * G =$$

$$= \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ s_{31} & s_{32} & s_{33} & s_{34} \end{pmatrix} + \frac{err}{8}\begin{pmatrix} 0 & 0(*) & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} =$$

$$= \begin{pmatrix} s_{11} + \frac{err}{8} \cdot 0 & s_{12} + \frac{err}{8} \cdot 0 & s_{13} + \frac{err}{8} \cdot 1 & s_{14} + \frac{err}{8} \cdot 1 \\ s_{21} + \frac{err}{8} \cdot 1 & s_{22} + \frac{err}{8} \cdot 1 & s_{23} + \frac{err}{8} \cdot 1 & s_{24} + \frac{err}{8} \cdot 0 \\ s_{31} + \frac{err}{8} \cdot 0 & s_{32} + \frac{err}{8} \cdot 1 & s_{33} + \frac{err}{8} \cdot 0 & s_{34} + \frac{err}{8} \cdot 0 \end{pmatrix}. \tag{2.16}$$

Аналогічно, отриманий результат (2.16) можна записувати не лише в матричному вигляді.

2.1.5. Алгоритм Беркеса

Основний підхід в алгоритмі Беркеса відбувається так. Для кожного пікселя накопичується помилка від його сусідів, причому дана помилка є зваженою, матриця ваг подається у вигляді:

$$G = \frac{1}{32}\begin{pmatrix} 0 & 0 & 0(*) & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \end{pmatrix}. \tag{2.17}$$

Матриця (2.17) накладається на центральний піксель, який позначено знаком зірочки «*». Для початку роботи алгоритму задається палітра кольорів, на основі якої буде формуватися нове зображення. У такому випадку для поточного пікселя, який обробляється знаходиться найближчий колір $NewColor$ з тієї палітри, на якій буде будуватись вихідне зображення:

$$err = s_{ij} - NewColor, \tag{2.18}$$

Помилка у (2.18) поширюється на сусідні пікселі на основі (2.1) за таким принципом:

Візьмемо фрагмент зображення, що записується у вигляді:

$$I = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \end{pmatrix}. \tag{2.19}$$

Слід зазначити, що у (2.19) значення s_{ij} s_{ij} $i \in \overline{1;2}, j \in \overline{1;5}$ можуть приймати цілі значення від 0 до 255 або значення від 0 до 1. На основі даних двох матриць знаходиться поелементний добуток, результат якого є оновлені значення інтенсивності пікселів за даним алгоритмом. Тобто маємо такий вираз:

$$\begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \end{pmatrix} = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \end{pmatrix} + err * G =$$
$$= \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \end{pmatrix} + \frac{err}{32}\begin{pmatrix} 0 & 0 & 0(*) & 8 & 4 \\ 2 & 4 & 8 & 4 & 2 \end{pmatrix} = \tag{2.20}$$
$$= \begin{pmatrix} s_{11} + \frac{err}{32} \cdot 0 & s_{12} + \frac{err}{42} \cdot 0 & s_{13} + \frac{err}{42} \cdot 0 & s_{14} + \frac{err}{42} \cdot 8 & s_{15} + \frac{err}{42} \cdot 4 \\ s_{21} + \frac{err}{32} \cdot 2 & s_{22} + \frac{err}{42} \cdot 4 & s_{23} + \frac{err}{42} \cdot 8 & s_{24} + \frac{err}{42} \cdot 4 & s_{25} + \frac{err}{42} \cdot 2 \end{pmatrix}.$$

2.1.6. Алгоритм Сієрри

Основний підхід в алгоритмі Сієрри передбачає, що для кожного пікселя накопичується помилка від його сусідів, причому дана помилка є зваженою, матриця ваг подається у вигляді:

$$G = \frac{1}{32}\begin{pmatrix} 0 & 0 & 0(*) & 5 & 3 \\ 2 & 4 & 5 & 4 & 2 \\ 0 & 2 & 3 & 2 & 0 \end{pmatrix}. \tag{2.21}$$

Матриця (2.21) накладається на центральний піксель, який позначено знаком зірочки «*». Для початку роботи алгоритму задається палітра кольорів, на основі якої буде формуватися нове зображення. У такому випадку для поточного пікселя, який обробляється знаходиться найближчий колір $NewColor$ з тієї палітри, на якій буде будуватись вихідне зображення:

$$err = s_{ij} - NewColor, \tag{2.22}$$

Помилка у (2.22) поширюється на сусідні пікселі на основі (2.1) за таким принципом:

Візьмемо фрагмент зображення, що записується у вигляді:

$$I = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix}. \tag{2.23}$$

Слід зазначити, що у (2.23) значення s_{ij} s_{ij} $i \in \overline{1;3}, j \in \overline{1;5}$ можуть приймати цілі значення від 0 до 255 або значення від 0 до 1. На основі даних двох матриць знаходиться поелементний добуток, результат якого є оновлені значення інтенсивності пікселів за даним алгоритмом. Тобто маємо такий вираз:

$$\begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} + err * G = \\ = \begin{pmatrix} s_{11} & s_{12} & s_{13} & s_{14} & s_{15} \\ s_{21} & s_{22} & s_{23} & s_{24} & s_{25} \\ s_{31} & s_{32} & s_{33} & s_{34} & s_{35} \end{pmatrix} + \frac{err}{32}\begin{pmatrix} 0 & 0 & 0(*) & 5 & 3 \\ 2 & 4 & 5 & 4 & 2 \\ 0 & 2 & 3 & 2 & 0 \end{pmatrix} = \\ = \begin{pmatrix} s_{11} + \frac{err}{32} \cdot 0 & s_{12} + \frac{err}{32} \cdot 0 & s_{13} + \frac{err}{32} \cdot 0 & s_{14} + \frac{err}{32} \cdot 5 & s_{15} + \frac{err}{32} \cdot 3 \\ s_{21} + \frac{err}{32} \cdot 2 & s_{22} + \frac{err}{32} \cdot 4 & s_{23} + \frac{err}{32} \cdot 5 & s_{24} + \frac{err}{32} \cdot 4 & s_{25} + \frac{err}{32} \cdot 2 \\ s_{31} + \frac{err}{32} \cdot 0 & s_{32} + \frac{err}{32} \cdot 2 & s_{33} + \frac{err}{32} \cdot 3 & s_{34} + \frac{err}{32} \cdot 2 & s_{35} + \frac{err}{32} \cdot 0 \end{pmatrix}. \tag{2.24}$$

2.1.7. Алгоритм Байєра

Метод Байєра дуже широко використовується, і його легко ідентифікувати за артефактами шаблону штрихування, які він створює на кінцевому дисплеї. Цей артефакт є головним недоліком техніки, яка в іншому випадку є дуже швидкою та потужною. Упорядковане згладжування також дуже погано працює на зображеннях , які вже були певною мірою згладжені, тому для таких зображень згладжування має бути останнім етапом у створенні фізичного дисплея із цифрового зображення.

Алгоритм зменшує кількість кольорів, застосовуючи порогову карту M до відображених пікселів, змушуючи деякі пікселі змінювати колір залежно від відстані вихідного кольору від доступних елементів кольорів у зменшеній палітрі. Порогові карти бувають різних розмірів, які зазвичай дорівнюють:

$$M_2 = \frac{1}{4}\begin{pmatrix} 0 & 2 \\ 3 & 1 \end{pmatrix};\ M_4 = \frac{1}{16}\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix};$$

$$M_8 = \frac{1}{64}\begin{pmatrix} 0 & 32 & 8 & 40 & 2 & 34 & 10 & 42 \\ 48 & 16 & 56 & 24 & 50 & 18 & 58 & 26 \\ 12 & 44 & 4 & 36 & 14 & 46 & 6 & 38 \\ 60 & 28 & 52 & 20 & 62 & 30 & 54 & 22 \\ 3 & 35 & 11 & 43 & 1 & 33 & 9 & 41 \\ 51 & 19 & 59 & 27 & 49 & 17 & 57 & 25 \\ 15 & 47 & 7 & 39 & 13 & 45 & 5 & 37 \\ 63 & 31 & 55 & 23 & 61 & 29 & 53 & 21 \end{pmatrix}.$$

Карту можна обертати або дзеркально відображати без впливу на ефективність алгоритму. Ця порогова карта (для сторін із довжиною у степені 2) також відома як індексна матриця або матриця Байєра [8]. Порогові карти довільного розміру можна розробити за простим правилом: спочатку заповніть кожен слот послідовним цілим числом. Потім змініть їх порядок таким чином, щоб середня відстань між двома послідовними числами на карті була якомога більшою, гарантуючи, що таблиця «загортається» по краях. Для порогових карт, розміри яких є ступенем двох, карту можна створити рекурсивно за допомогою:

$$M_{2n} = \frac{1}{(2n)^2}\begin{pmatrix} (2n)^2 M_n & (2n)^2 M_n + 2 \\ (2n)^2 M_n + 3 & (2n)^2 M_n + 1 \end{pmatrix}.$$

Ця функція також може бути виражена за допомогою лише бітової арифметики.

2.2. Алгоритми випадкового дизерингу

Випадковий дизеринг включає декілька підходів, які використовують різні методи додавання випадкового шуму до пікселів зображення перед їх квантуванням. Існує кілька алгоритмів випадкового дизерингу.

2.2.1. Простий випадковий дизеринг

Цей метод використовує випадковий шум для кожного пікселя, щоб вирішити, до якого кольору його прив'язати [12].

Алгоритм простого випадкового дизерингу з поширенням похибки

1) Додавання випадкового шуму та обчислення помилки:
   1.1) $\eta \sim U(-d, d)$.
   1.2) $I'(x, y) = I(x, y) + \eta$.
   1.3) $I_{new}(x, y) = closest(I'(x, y), palette)$.
   1.4) $\varepsilon = I'(x, y) - I_{new}(x, y)$.
2) Розподіл помилки (можна використовувати вагові коефіцієнти як у алгоритмі Флойда-Штейнберга):
   2.1) $I(x + 1, y) \leftarrow I(x + 1, y) + \varepsilon \cdot 7/16$.
   2.2) $I(x - 1, y + 1) \leftarrow I(x - 1, y + 1) + \varepsilon \cdot 3/16$.
   2.3) $I(x, y + 1) \leftarrow I(x, y + 1) + \varepsilon \cdot 5/16$.
   2.4) $I(x + 1, y + 1) \leftarrow I(x + 1, y + 1) + \varepsilon \cdot 1/16$.

Такий алгоритм можна виконувати в матричній формі, як було подано усі алгоритми детермінованого дизерингу.

2.2.2. Рівномірний випадковий дизеринг

Цей алгоритм передбачає додавання білого шуму (рівномірний розподіл) до пікселів зображення перед їх квантуванням. Алгоритм рівномірного випадкового дизерингу:

1) Додавання білого шуму

$$\eta \sim U(-\alpha, \alpha),$$

де α – амплітуда шуму.

2) Виконання модифікації інтенсивності пікселя:

$$I'(x, y) = I(x, y) + \eta.$$

3) Здійснення вибору найближчого кольору:

$$I_{new}(x, y) = closest(I'(x, y), palette).$$

2.2.3. Гауссовий випадковий дизеринг

Цей алгоритм додає гауссовий шум (нормальний розподіл) до пікселів зображення перед їх квантуванням. Алгоритм гауссового випадкового дизерингу передбачає:

1) Додавання гауссового шуму

$$\eta \sim N(0, \sigma),$$

де N – нормальний розподіл, σ – стандартне відхилення.

2) Виконання модифікації інтенсивності пікселя:

$$I'(x, y) = I(x, y) + \eta.$$

3) Здійснення вибору найближчого кольору:

$$I_{new}(x, y) = closest(I'(x, y), palette).$$

2.2.4. Випадковий упорядкований дизеринг

Метод упорядкованого випадкового дизерингу використовує випадкові матриці для дизерингу, замість регулярних матриць, таких як Байєрова матриця. Алгоритм упорядкованого випадкового дизерингу:

1) Створюється випадкова матриця T розміром $n \times n$.
2) Додається випадковий шуму, який береться з матриці T.
3) Виконується модифікація інтенсивності пікселя:

$$I'(x, y) = I(x, y) + T(x \bmod n, y \bmod n).$$

4) Здійснюється вибір найближчого кольору:

$$I_{new}(x, y) = closest(I'(x, y), palette).$$

2.3. Особливості програмної реалізації алгоритмів

2.3.1. Вибір мови та інструментів розроблення програмного забезпечення реалізації алгоритмів

Обробка зображень є однією з важливих задач у багатьох областях, включаючи комп'ютерний зір, графіку, медичну візуалізацію, дизеринг (алгоритми якого описані в роботі) тощо. Існує багато мов програмування, які можуть використовуватися для обробки зображень. Кожна з них має свої переваги та недоліки. Нижче наведено порівняльний аналіз кількох популярних мов програмування для обробки зображень.

Мова програмування Python. У таблиці 2.1 наведені основні переваги і недоліки мови програмування Python при обробленні зображень, зокрема у задачах дизерингу.

Таблиця 2.1 – Переваги і недоліки Python в задачах цифрового опрацювання зображень та відео

Переваги	Недоліки
✓ Наявність чудового набору бібліотек OpenCV, PIL/Pillow, scikit-image, NumPy. ✓ Легка для навчання та використання, особливо для новачків. ✓ Велике співтовариство користувачів, багато прикладів та документації. ✓ Легко інтегрується з іншими науковими бібліотеками, такими як TensorFlow, Keras, та PyTorch для машинного навчання.	✓ Може бути повільнішим порівняно з C++ або C. ✓ Вимагає більше пам'яті для великих обчислень.

Мова програмування C++. У таблиці 2.2 наведені основні переваги і недоліки мови програмування C++ при обробленні зображень, зокрема у задачах дизерингу.

Таблиця 2.2 – Переваги і недоліки Python в задачах цифрового опрацювання зображень та відео

Переваги	Недоліки
✓ Оптимізований для швидкого виконання та ефективного використання пам'яті. ✓ Потужні бібліотеки для обробки зображень, такі як OpenCV. ✓ Дає повний контроль над управлінням пам'яттю та апаратним забезпеченням.	✓ Більш складний синтаксис та управління пам'яттю порівняно з Python. ✓ Менше готових прикладів та документації для новачків.

Система комп'ютерної математики MATLAB. У таблиці 2.3 наведені основні переваги і недоліки MATLAB при обробленні зображень, зокрема у задачах дизерингу.

Таблиця 2.3 – Переваги і недоліки MATLAB в задачах цифрового опрацювання зображень та відео

Переваги	Недоліки
✓ Ідеальний для швидкого прототипування та візуалізації. ✓ Вбудовані функції для обробки зображень та аналізу. ✓ Велика кількість офіційної документації та підтримка з боку MathWorks.	✓ MATLAB є комерційним продуктом, що потребує купівлі ліцензії. ✓ Може бути повільнішим порівняно з мовами на кшталт C++ для великих задач.

Мова програмування Java. У таблиці 2.4 наведені основні переваги і недоліки мови програмування Java при обробленні зображень, зокрема у задачах дизерингу.

Таблиця 2.4 – Переваги і недоліки Java в задачах цифрового опрацювання зображень та відео

Переваги	Недоліки
✓ Працює на будь-якій платформі з встановленою Java Virtual Machine (JVM). ✓ Потужні бібліотеки для обробки зображень, такі як ImageJ, BoofCV. ✓ Вбудовані засоби для управління пам'яттю та виключеннями.	✓ Повільніша за C++ через JVM. ✓ Може бути складнішою у використанні для новачків порівняно з Python.

Мова програмування C#. У таблиці 2.5 наведені основні переваги і недоліки мови програмування C# при обробленні зображень, зокрема у задачах дизерингу.

Таблиця 2.5 – Переваги і недоліки C# в задачах цифрового опрацювання зображень та відео

Переваги	Недоліки
✓ Ідеальний для розробки додатків під Windows. ✓ Потужні бібліотеки, такі як AForge.NET, Accord.NET для обробки зображень. ✓ Легка інтеграція з іншими .NET бібліотеками та інструментами.	✓ Менш підходить для розробки під інші платформи, хоча існує підтримка через .NET Core. ✓ Менше прикладів та підтримки для обробки зображень порівняно з Python.

Мова програмування Julia. У таблиці 2.6 наведені основні переваги і недоліки мови програмування Julia при обробленні зображень, зокрема у задачах дизерингу.

Таблиця 2.6 – Переваги і недоліки Julia в задачах цифрового опрацювання зображень та відео

Переваги	Недоліки
✓ Порівняна з продуктивністю C++.. ✓ Існують бібліотеки для обробки зображень, такі як Images.jl. ✓ Простіша у використанні порівняно з C++.	✓ Менше бібліотек та спільноти порівняно з Python або C++. ✓ Ще не така зріла як інші мови, менше готових прикладів та документації.

Мова програмування R. У таблиці 2.7 наведені основні переваги і недоліки мови програмування R при обробленні зображень, зокрема у задачах дизерингу.

Таблиця 2.7 – Переваги і недоліки R в задачах опрацювання зображень та відео

Переваги	Недоліки
✓ Потужний для статистичного аналізу зображень. ✓ Є спеціалізовані бібліотеки для обробки зображень, такі як imager.	✓ Менше бібліотек та спільноти порівняно з Python або C++. ✓ Повільніший порівняно з мовами на кшталт C++.

Слід також зазначити, що вибір мови програмування для обробки зображень залежить від конкретних вимог проєкту, включаючи продуктивність, простоту використання, наявність бібліотек та інші фактори. Кожна мова має свої сильні та слабкі сторони, і вибір залежить від конкретного завдання. Але, у зв’язку з простотою мови програмування Python та її доступності, вибір було здійснено на її користь.

2.3.2. Структурна схема програмного забезпечення

Розроблене програмне забезпечення складається з кількох програмних модулів, які умовно можна поділити на такі категорії:

1) модулі завантаження і збереження даних. Ці модулі передбачають роботу з самим зображенням та результатом оброблення зображення одним з тих алгоритмів, які наведені у даному розділі;

2) модулі реалізації детермінованих алгоритмів цифрового дизерингу. У них безпосередньо занесений код, який виконує опрацювання зображень за конкретним детермінованим алгоритмом;
3) модулі реалізації стохастичних алгоритмів цифрового дизерингу. У них безпосередньо занесений код, який виконує опрацювання зображень за конкретним стохастичним алгоритмом;
4) модулі порівняння роботи алгоритмів за критерієм часу, пікового відношення сигналу до шуму. Дані модулі виконують неодноразовий запуск модулів з алгоритмами для різних вхідних даних (зображення різної розмірності).

Схематично можна навести структуру програмного забезпечення (рисунок 2.1). Короткий опис деяких модулів буде описано далі.

2.3.3. Критерії оцінки якості роботи алгоритмів

Середньоквадратична помилка (MSE) – це показник середнього квадратичного відхилення між передбаченими значеннями моделі та фактичними значеннями. Він широко використовується в статистиці, регресійному аналізі та машинному навчанні, а також у ході аналізу зображень у цифровій формі, зокрема, у задачах дизерингу.

Середньоквадратична помилка обчислюється за формулою:

$$MSE = \frac{1}{M \cdot N} \sum_{x=1}^{M} \sum_{y=1}^{N} \left(I(x,y) - I'(x,y) \right)^2.$$

В останній формулі є такі компоненти:

- I – вхідне зображення;
- $I(x,y)$ – піксель з координатами *x, y* вхідного зображення I;
- I' – вихідне зображення;
- $I'(x,y)$ – піксель з координатами *x, y* вхідного зображення I';
- M – кількість рядків з пікселями у кожному з зображень;
- N – кількість пікселів у кожному рядку в кожному з зображень.

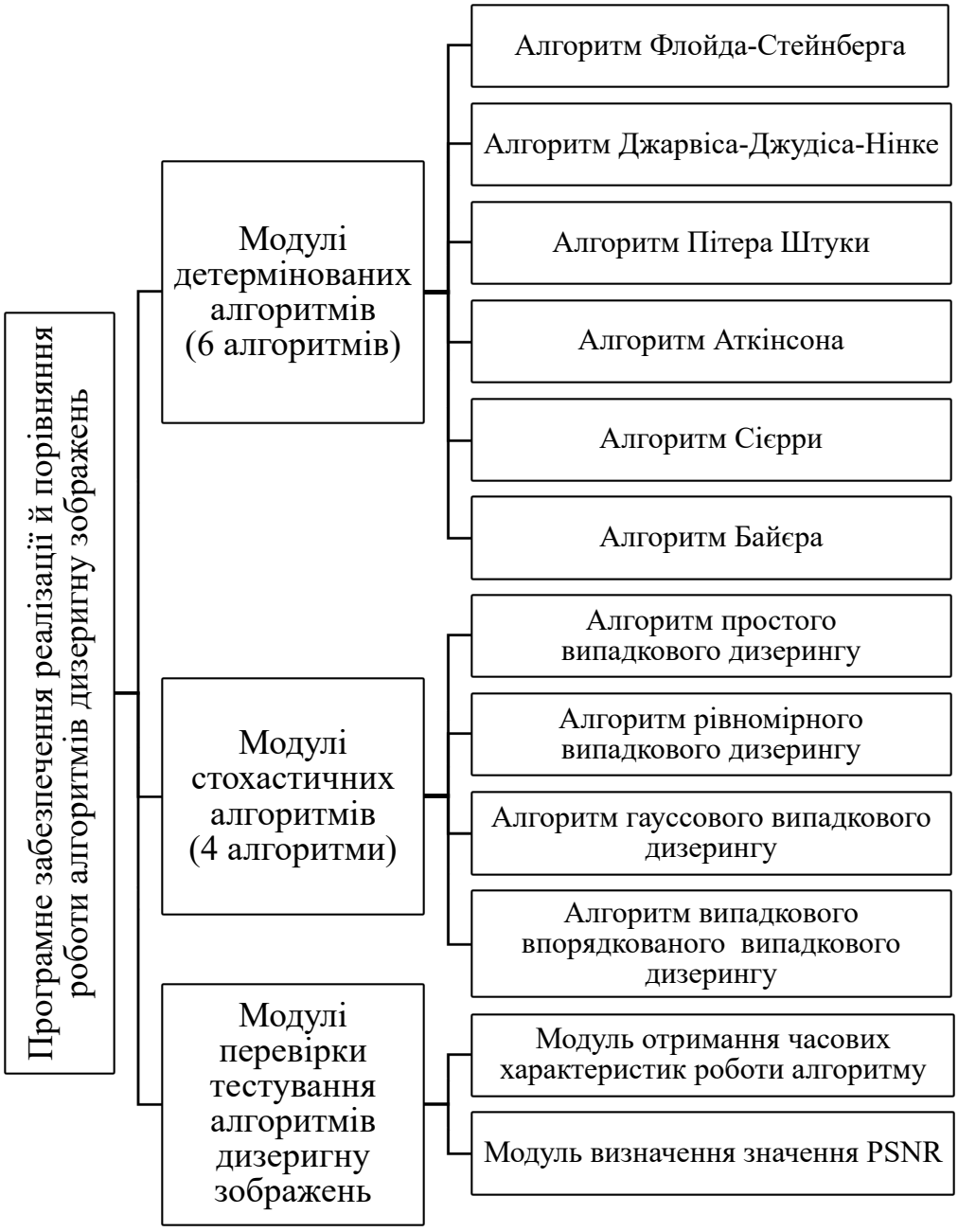


Рисунок 2.1 – Структура розробленого програмного застосунку порівняння алгоритмів дизерингу зображень

MSE часто використовується в задачах регресії, де важливо кількісно оцінити різницю між передбаченнями моделі та фактичними значеннями, а також у налаштуванні моделей машинного навчання, таких як лінійна регресія, нейронні мережі та інші алгоритми. MSE є основним інструментом для оцінки якості моделі і її налаштування, допомагаючи зрозуміти, наскільки добре модель підходить для даних, і дозволяє порівнювати різні моделі між собою. Наступний

критерій оцінки якості дизерингу це пікове відношення сигналу до шуму. Пікове відношення сигналу до шуму (PSNR) – це міра якості відновленого або стиснутого зображення або відео порівняно з оригіналом. PSNR широко використовується в області обробки зображень та відео для оцінки якості стислих або відновлених зображень. Це логарифмічна міра, що виражається в децибелах (dB). Формула для PSNR виглядає так:

$$\text{PSNR} = 10 \cdot \log_{10}\left(\frac{MAX^2}{MSE}\right),$$

де *MAX* – максимальне можливе значення пікселя зображення. Для 8-бітного зображення це 255; MSE – середньоквадратична помилка між оригінальним та відновленим зображенням, формула обчислення якої була наведена вище.

Слід зазначити, що PSNR є важливим показником для оцінки якості зображень і відео після обробки. Високі значення PSNR вказують на високу якість, тоді як низькі значення свідчать про суттєві спотворення. Використання PSNR дозволяє об’єктивно порівнювати різні алгоритми обробки і стиснення зображень та відео.

2.3.4. Оптимізація програмного коду

З урахуванням того, що в якості основної мови розроблення програмного забезпечення було обрано саме Python, то оптимізація програмного коду на Python є актуальним питанням, яке може значно покращити продуктивність і ефективність розробленої програми. Для прискорення обчислень у розробленому програмному забезпеченні були використані відомі рішення проблеми повільних обчислень, а також бібліотека NumPy для обробки великих масивів даних. Ще однією з можливих методів прискорення обчислень є багатопоточність та мультипроцесинг. Усі алгоритми дизерингу можуть бути розпаралелені, але у роботі акцент був лише на порівняння послідовних алгоритмів, тобто в тому вигляді, у якому вони подані в цій роботі.

РОЗДІЛ 3
ПОРІВНЯЛЬНИЙ АНАЛІЗ АЛГОРИТМІВ ДИЗЕРИНГУ

3.1. Методика порівняння алгоритмів

Методика порівняння розглянутих у роботі алгоритмів дизерингу складається з кількох частин. Перша частина передбачає звичайне тестування методів на різних зображеннях, зокрема, на зображеннях різної розмірності. Критеріями тестування є час виконання програми, загальні витрати ресурсів на виконання програмного коду, що реалізує конкретний алгоритм дизерингу.

Окрім того, проводиться міжкласове порівняння алгоритмів за принципом, до якого типу відносять той або інший алгоритм. Тобто відбувається порівняння детермінованих алгоритмів і стохастичних за критеріями, які наведені вище.

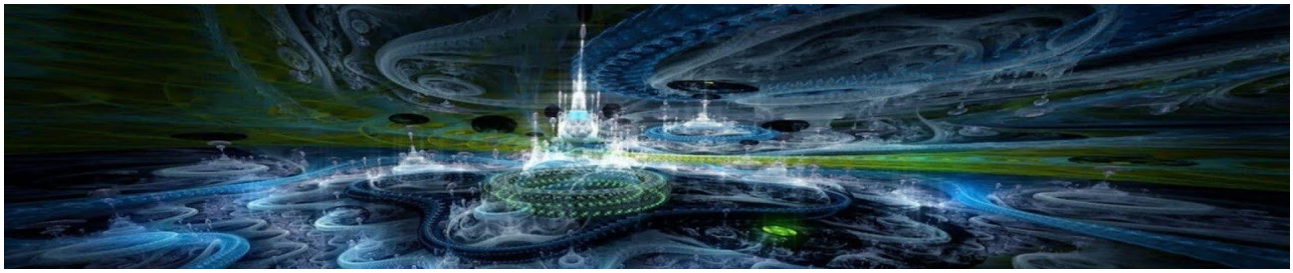


Рисунок 3.1 – Оригінал зображення

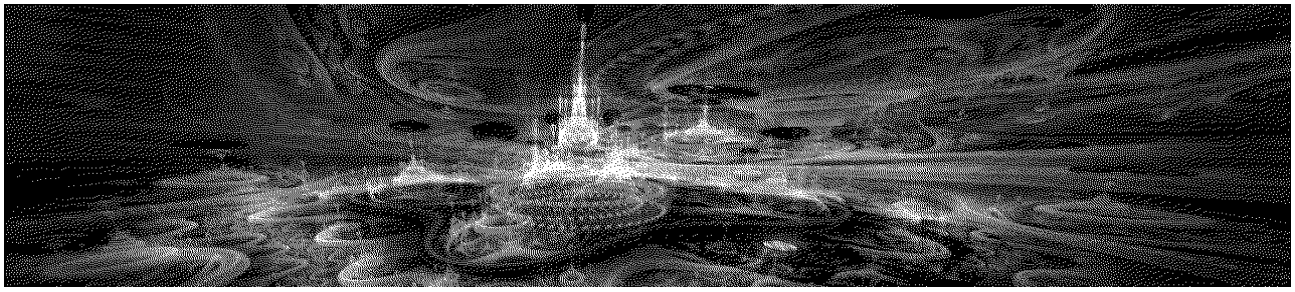


Рисунок 3.2 – Результат дизерингу за алгоритмом Флойда-Стейнберга для двох кольорів (чорний і білий)

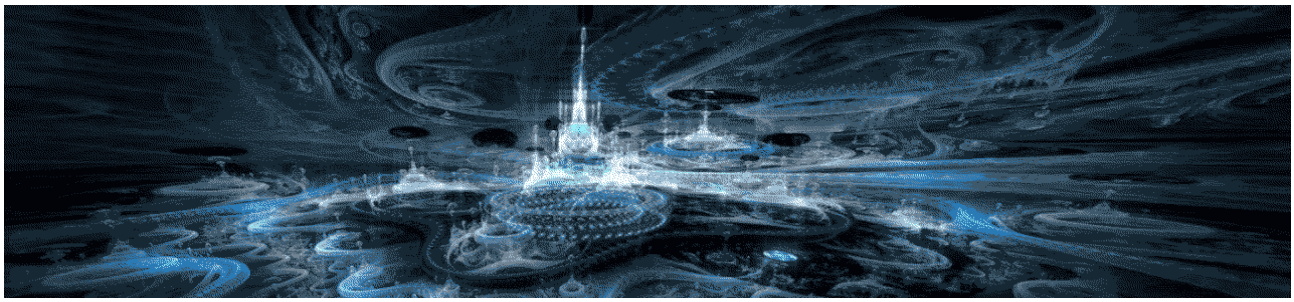


Рисунок 3.3 – Результат дизерингу за алгоритмом Флойда-Стейнберга для шести кольорів

На рисунках 3.1–3.20 наведені основні результати виконання наведених у попередньому розділі алгоритмів

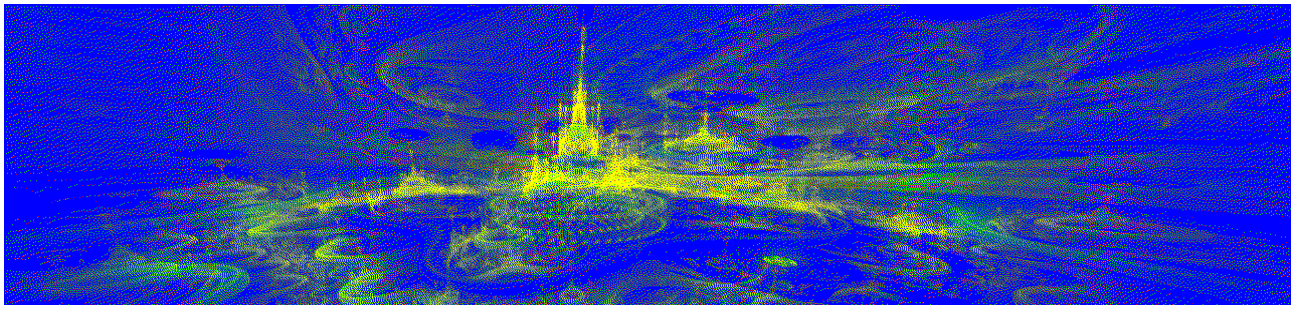


Рисунок 3.4 – Результат дизерингу за алгоритмом Флойда-Стейнберга для чотирьох довільних кольорів

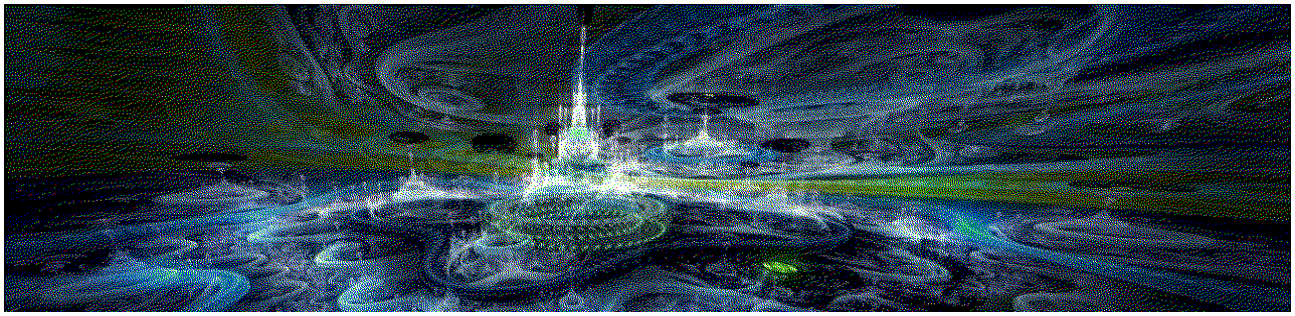


Рисунок 3.5 – Результат дизерингу за алгоритмом Флойда-Стейнберга для довільних кольорів (чорний, білий, зелений і синій)

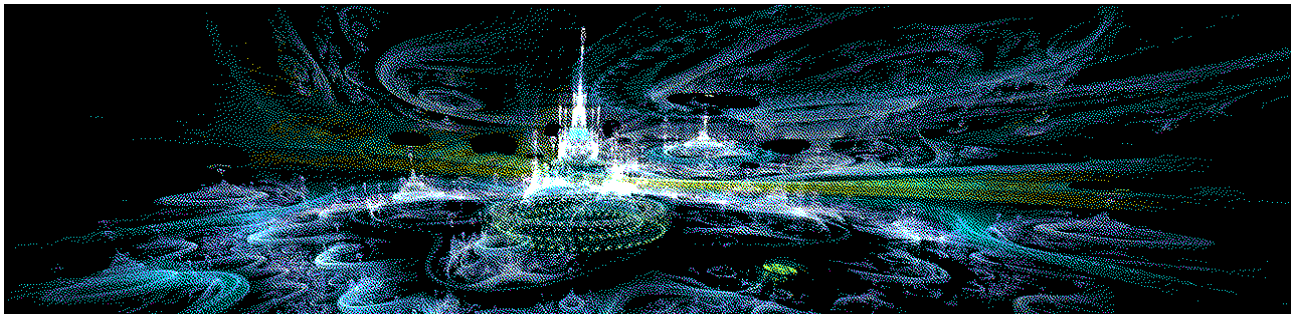


Рисунок 3.6 – Результат дизерингу за алгоритмом Аткінсона для чотирьох кольорів (чорний, білий, зелений і синій)

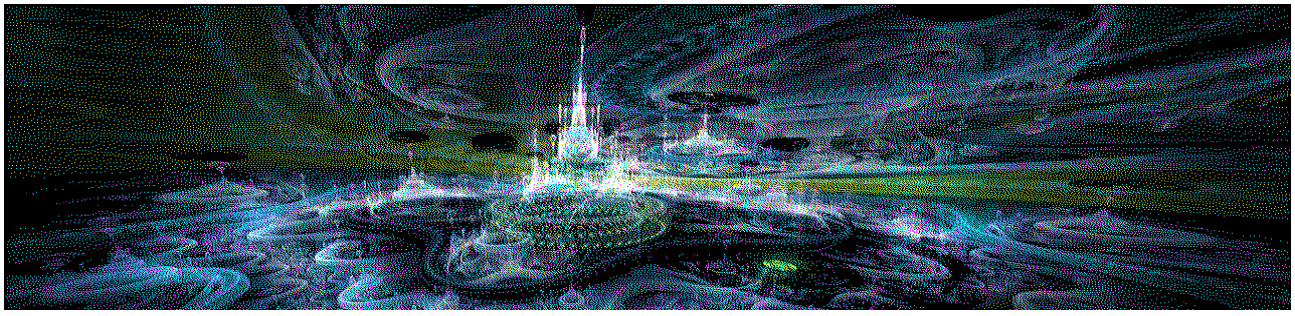


Рисунок 3.7 – Результат дизерингу за алгоритмом Беркеса для чотирьох кольорів (чорний, білий, зелений і синій)

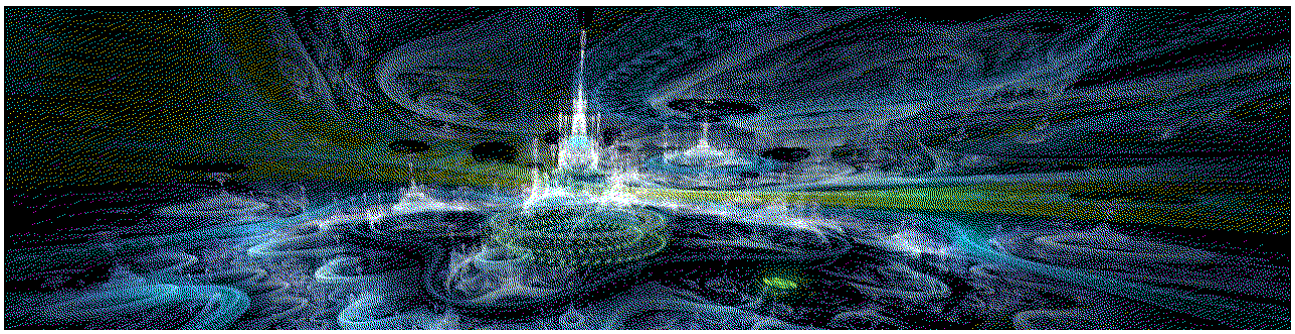


Рисунок 3.8 – Результат дизерингу за алгоритмом Сієрри для чотирьох кольорів (чорний, білий, зелений і синій)

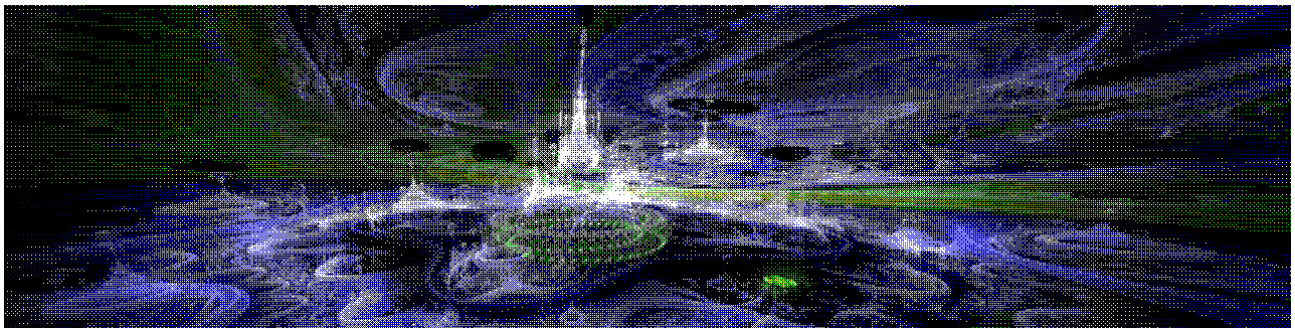


Рисунок 3.9 – Результат дизерингу за алгоритмом Байєра для чотирьох кольорів (чорний, білий, зелений і синій)

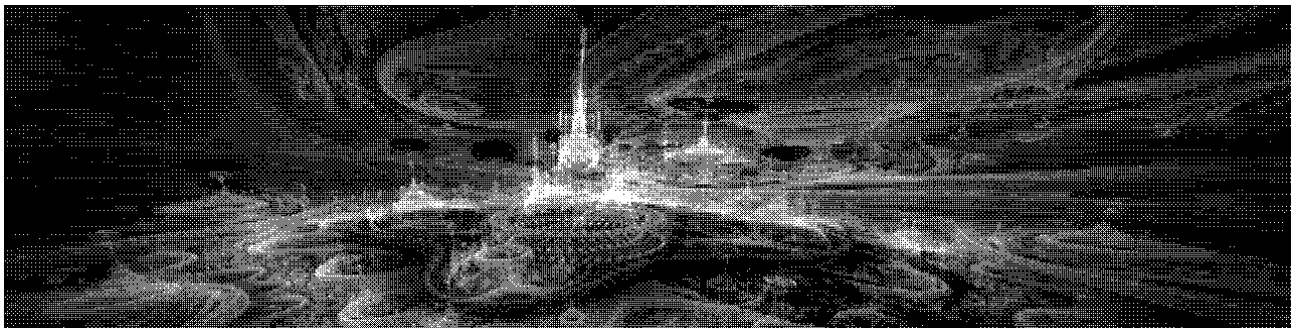


Рисунок 3.10 – Результат дизерингу за алгоритмом Байєра для двох кольорів (чорний і білий)

Аналогічна процедура буде проведена для прикладу для іншого зображення іншої розмірності і природи загалом.

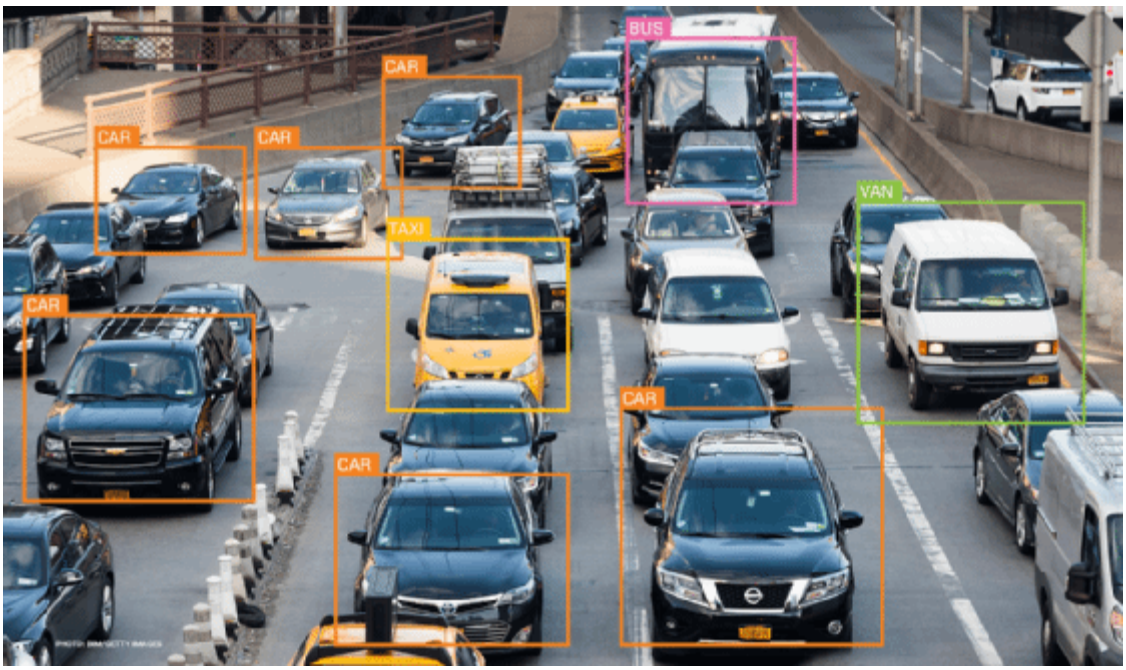


Рисунок 3.11 – Оригінальне зображення

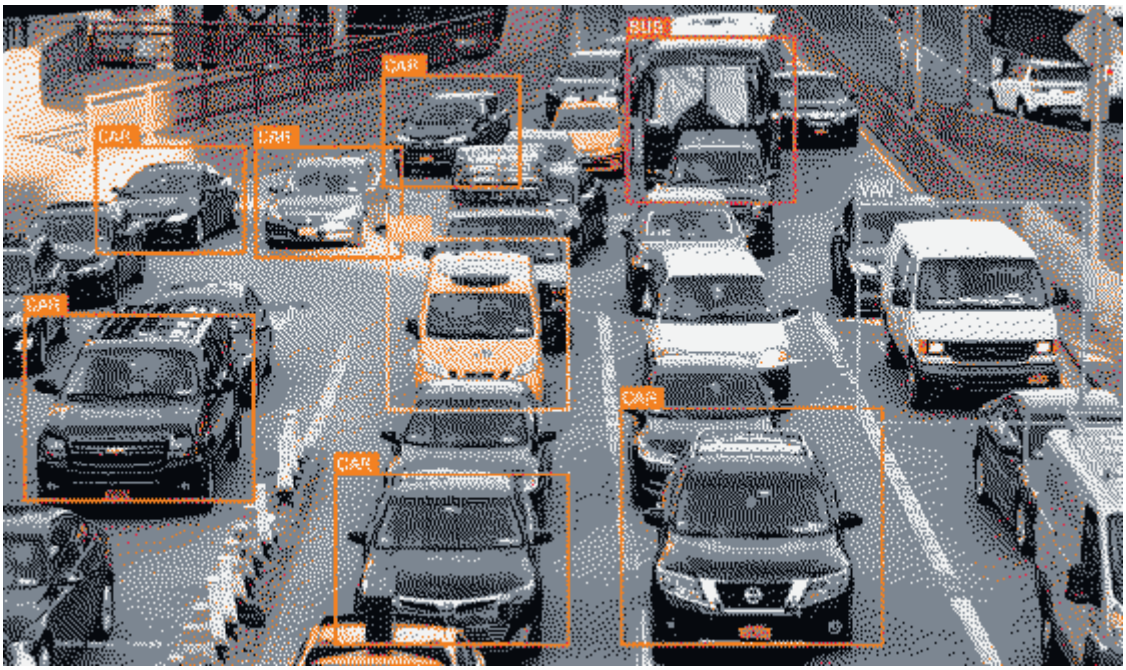


Рисунок 3.11 – Результат роботи алгоритму Флойда-Стейнберга для 4-х кольорів

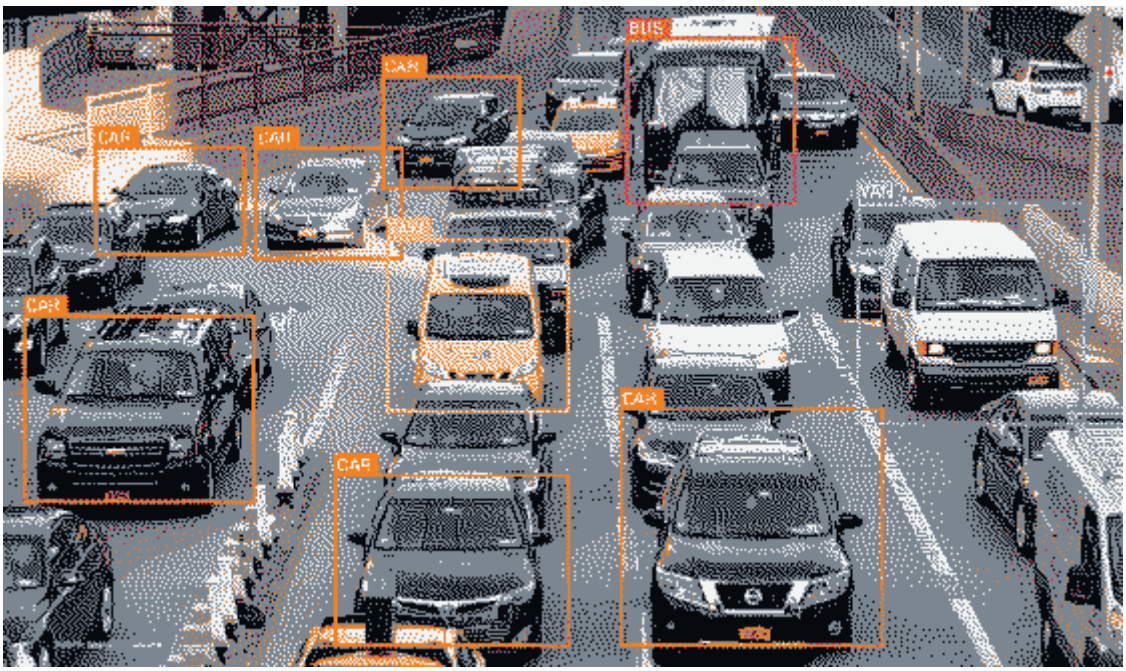


Рисунок 3.12 – Результат роботи алгоритму Беркеса для 4-х кольорів

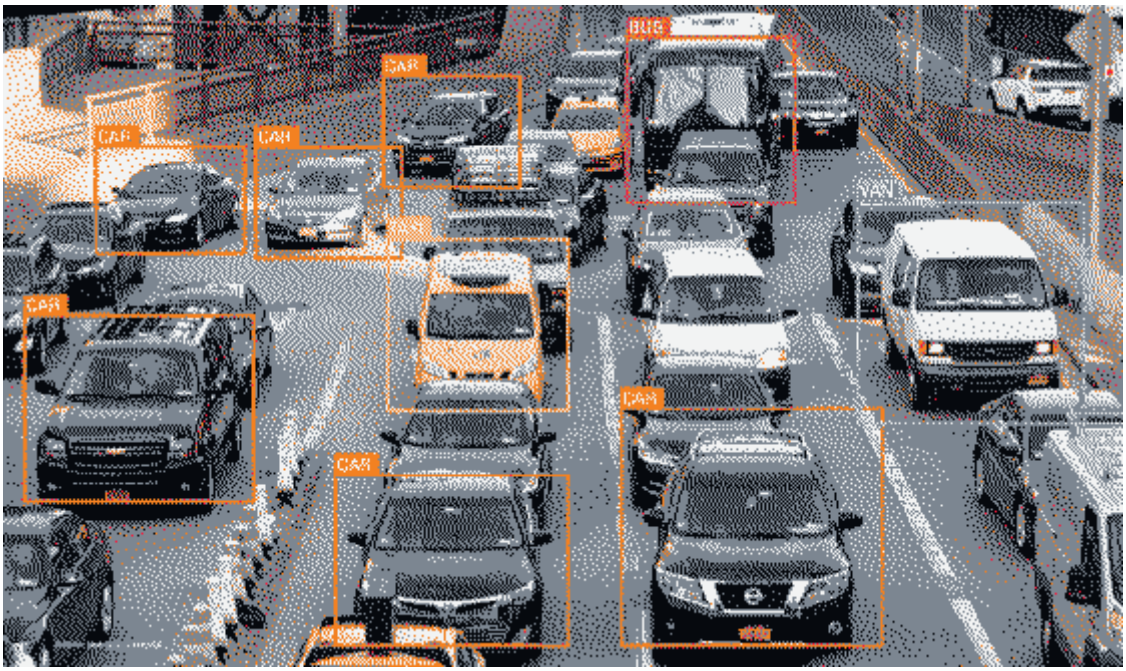


Рисунок 3.13 – Результат роботи алгоритму Пітера Штуки для 4-х кольорів

Рисунок 3.14 – Результат роботи алгоритму Байєера для 4-х кольорів

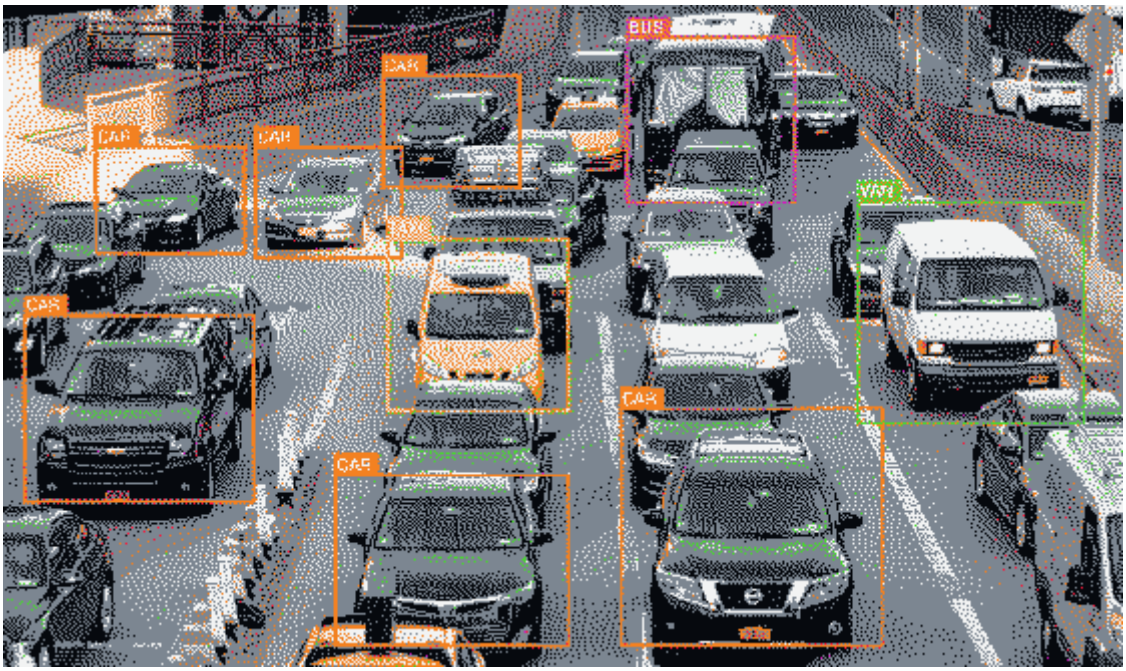


Рисунок 3.15 – Результат роботи алгоритму Сієрри для 4-х кольорів

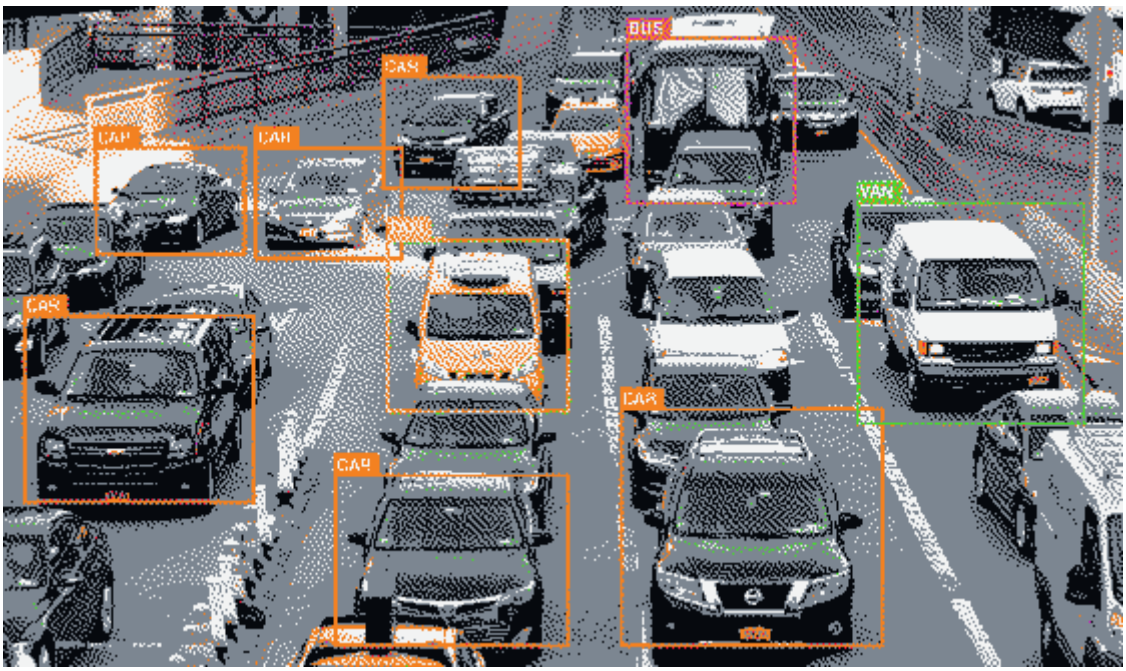


Рисунок 3.15 – Результат роботи алгоритму Аткінсона для 4-х кольорів

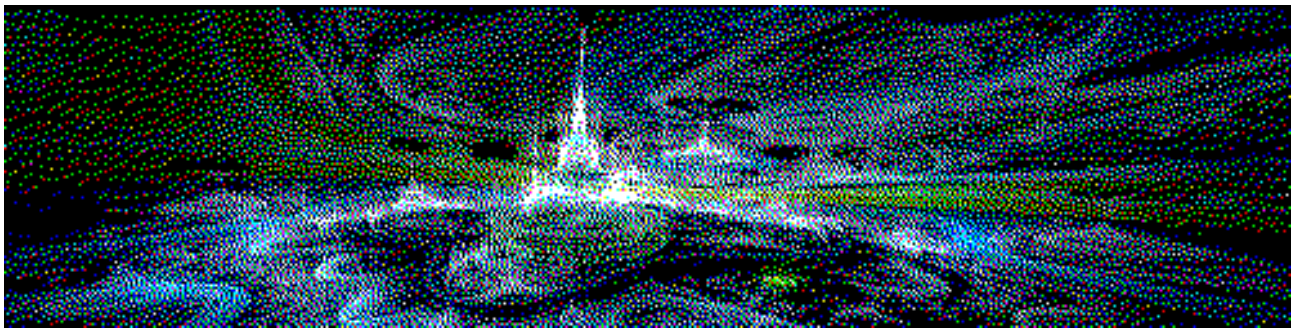


Рисунок 3.16 – Результат роботи простого випадкового дизерингу випадкових для 4-х кольорів

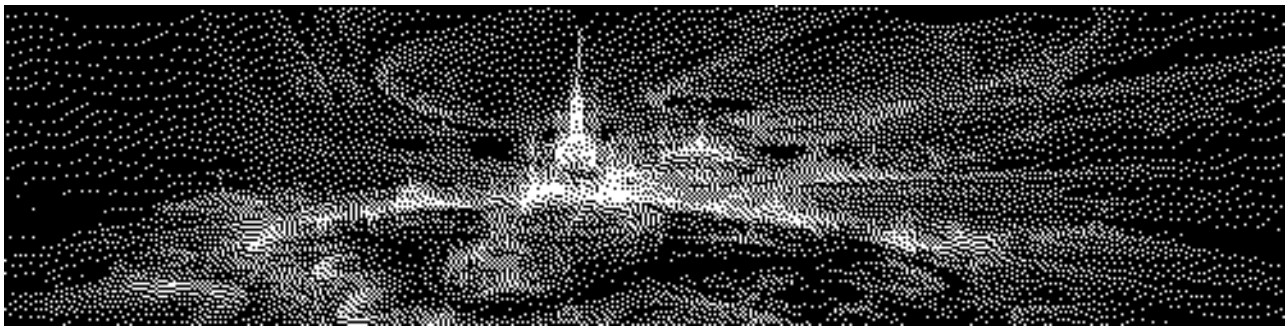


Рисунок 3.17 – Результат роботи простого випадкового дизерингу для 2-х кольорів (чорний і білий)

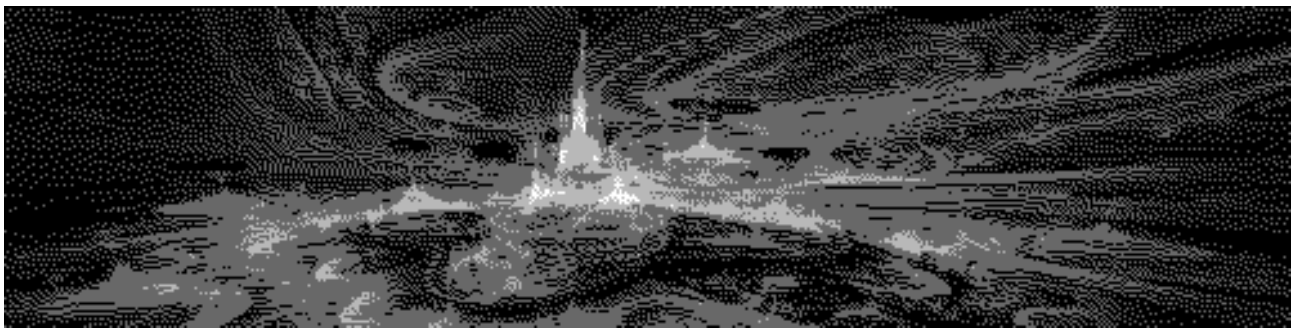


Рисунок 3.18 – Результат роботи випадкового рівномірного дизерингу для 2-х кольорів (чорний і білий)

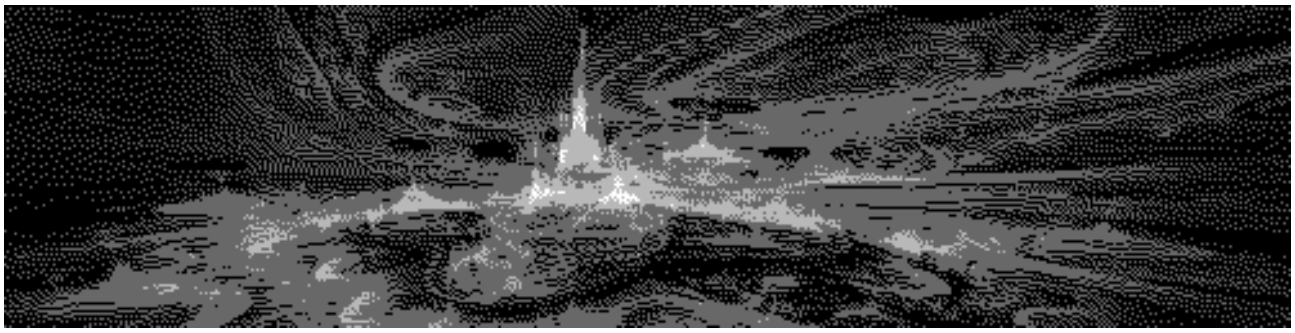


Рисунок 3.18 – Результат роботи випадкового рівномірного дизерингу для 2-х кольорів (чорний і білий)

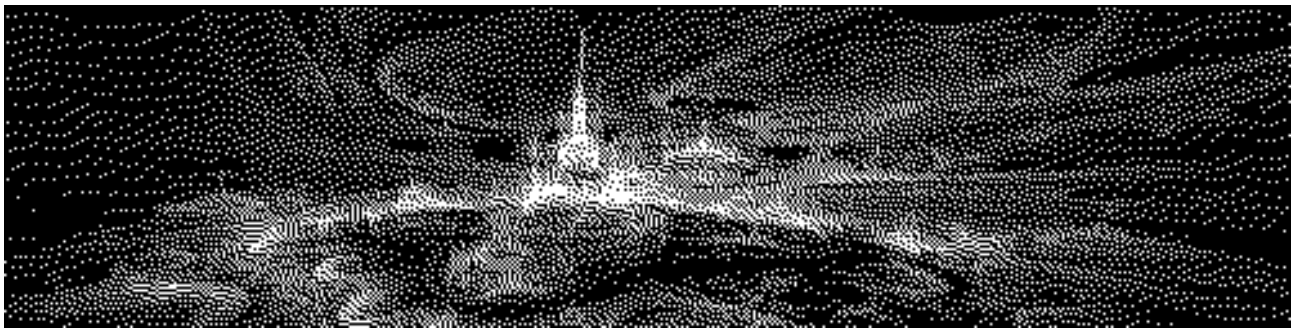


Рисунок 3.19 – Результат роботи випадкового гауссового дизерингу для 2-х кольорів (чорний і білий)

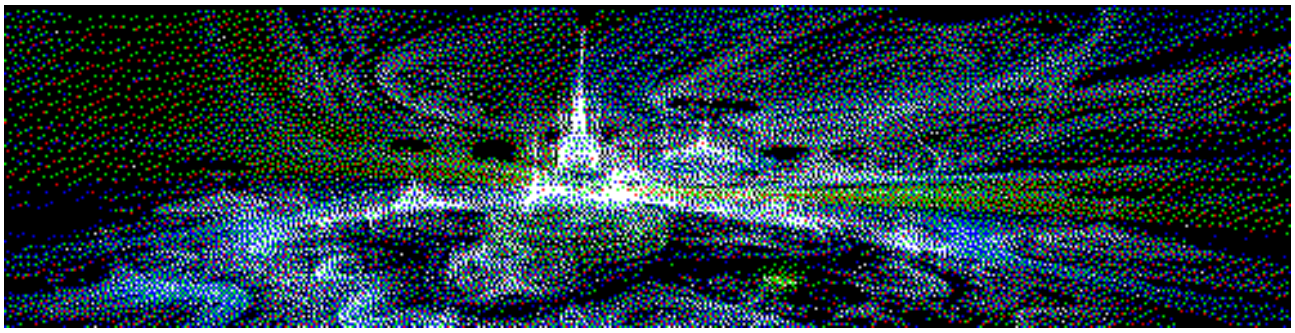


Рисунок 3.20 – Результат роботи випадкового упорядкованого дизерингу для 4 кольорів (чорний і білий, синій, заланий)

3.2. Порівняння детермінованих та стохастичних версій алгоритмів

Наведемо кілька таблиць, які демонструють ефективність кожного з розглянутих класів алгоритмів цифрового дизерингу.

Таблиця 3.1 відповідає порівняльному аналізу детермінованих розглянутих у роботі алгоритмів на основі PSNR. Таблиця 3.2 відповідає порівняльному аналізу стохастичних алгоритмів на основі PSNR.

Таблиця 3.1 – Порівняльний аналіз детермінованих алгоритмів, які описані у роботі, на основі результатів PSNR

				Алгоритми упорядкованого дизерингу									
Тестове зображення	Кількість рядків з пікселями, M	Кількість пікселів у кожному рядку, N	Загальна кількість пікселів	Алгоритм розсіювання помилок Флойда–Стейнберга	Алгоритм Джарвіса–Джуліса–Нінке	Алгоритм Пітер Штуки	Алгоритм Аткінсона	Алгоритм Беркеса	Алгоритм Сієрри	Алгоритм Байєра	Алгоритм Беркеса	Алгоритм Сієрри	Алгоритм Байєра
1	240	320	76800	60,135	65,228	60,616	60,321	60,077	60,680	60,031	60,810	60,155	55,040
2	288	320	92160	60,168	65,847	60,702	61,711	60,442	61,571	60,988	61,013	60,231	55,171
3	240	400	96000	60,589	66,301	60,759	62,271	60,790	62,579	61,362	61,048	60,538	55,677
4	480	576	276480	60,668	66,696	60,813	62,496	60,797	63,492	61,435	61,398	60,970	57,220
5	240	640	153600	60,923	67,716	61,837	62,692	61,089	64,228	61,677	62,164	61,408	58,820
6	320	480	153600	61,423	68,185	62,274	62,693	62,963	65,961	62,193	62,930	61,759	60,285
7	480	800	384000	62,160	68,442	62,477	64,065	63,236	66,447	63,164	63,261	61,964	60,838
8	480	854	409920	62,568	70,536	63,251	66,113	63,307	66,960	63,226	64,474	62,095	61,403
9	600	800	480000	63,887	71,075	63,532	66,834	63,999	67,263	65,282	64,826	63,271	61,726
10	540	960	518400	64,615	72,528	64,418	67,204	64,505	67,965	66,295	65,279	64,415	61,746
11	600	1066	639600	66,061	72,697	64,421	67,400	64,731	67,968	67,450	66,233	64,464	62,460
12	768	1024	786432	67,157	73,122	64,800	67,760	65,252	68,195	68,730	66,424	64,678	62,681
13	720	1280	921600	67,377	73,224	64,973	68,317	66,596	68,621	68,806	66,552	65,248	62,975
14	864	1152	995328	67,694	73,384	65,001	68,981	67,670	68,840	68,930	67,956	65,752	64,600
15	900	1440	1296000	68,396	74,231	65,291	69,045	68,154	68,937	68,969	68,655	66,382	64,719
16	1024	1280	1310720	69,373	75,041	65,935	69,647	68,650	69,175	69,700	68,987	68,257	64,725
17	1050	1440	1512000	69,397	75,161	66,122	69,787	69,009	69,389	69,722	69,017	68,279	65,534
18	1200	1600	1920000	70,166	75,677	69,579	70,601	69,442	70,504	70,100	69,352	70,381	65,540
19	1024	2048	2097152	70,209	75,847	70,279	70,772	69,882	70,639	70,560	69,933	70,432	65,828
20	2048	3200	6553600	70,443	75,880	70,780	70,826	70,887	70,990	70,612	70,041	70,794	62,335

Графічно результати таблиці 3.1 можна показати у вигляді рисунка 3.21.

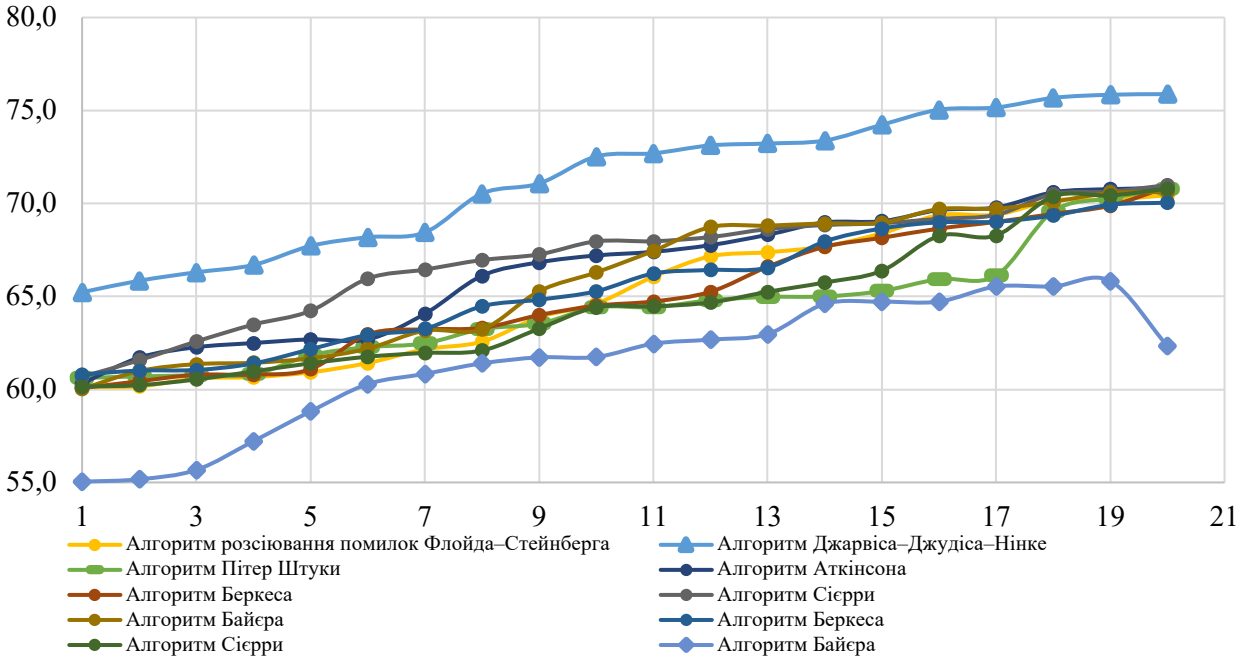


Рисунок 3.21 – Порівняльний аналіз детермінованих алгоритмів дизерингу на основі отриманих даних PSNR (вісь абсцис – номер тестового зображення з таблиці 3.1, вісь ординат – значення отриманого PSNR-показника за результатами алгоритмів)

Таблиця 3.2 – Порівняльний аналіз детермінованих алгоритмів, які описані у роботі, на основі результатів PSNR

				Алгоритми випадкового дизерингу			
Тестове зображення	Кількість рядків з пікселями, M	Кількість пікселів у кожному рядку, N	Загальна кількість пікселів	Простий випадковий дизеринг	Рівномірний випадковий дизеринг	Гауссовий випадковий дизеринг	Випадковий упорядкований дизеринг
1	240	320	76800	45,378	45,305	45,336	46,422
2	288	320	92160	46,842	45,546	45,877	46,734
3	240	400	96000	46,850	46,021	46,011	47,432
4	480	576	276480	46,880	46,379	46,519	48,880
5	240	640	153600	46,943	46,448	48,034	49,174
6	320	480	153600	46,985	47,506	48,626	49,468
7	480	800	384000	47,416	48,030	49,031	49,661
8	480	854	409920	47,970	48,307	49,514	50,107
9	600	800	480000	48,680	49,046	50,024	50,594
10	540	960	518400	49,259	49,728	50,087	50,651
11	600	1066	639600	49,632	50,894	50,413	50,903
12	768	1024	786432	50,449	51,485	51,129	52,124
13	720	1280	921600	51,976	51,862	52,777	52,235
14	864	1152	995328	52,012	52,078	53,169	52,854
15	900	1440	1296000	52,057	52,188	54,113	53,322
16	1024	1280	1310720	54,768	52,806	54,414	53,749
17	1050	1440	1512000	55,174	53,516	55,008	53,892
18	1200	1600	1920000	55,222	54,743	55,350	53,935
19	1024	2048	2097152	55,408	55,354	55,691	54,465
20	2048	3200	6553600	55,897	55,743	55,882	55,496

Графічно результати таблиці 3.2 можна показати у вигляді рисунка 3.22.

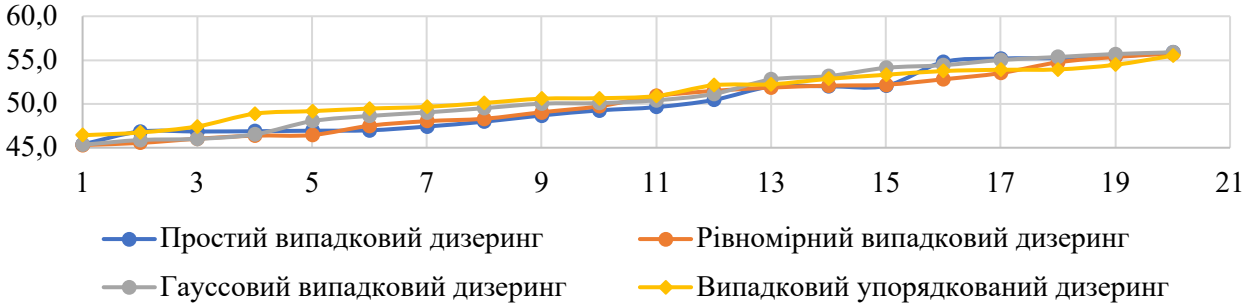


Рисунок 3.22 – Порівняльний аналіз детермінованих алгоритмів дизерингу на основі отриманих даних PSNR (вісь абсцис – номер тестового зображення з таблиці 3.2, вісь ординат – значення отриманого PSNR-показника за результатами алгоритмів)

3.3. Порівняння продуктивності та витрат ресурсів

У даному підрозділі наведені результати тестування методів упорядкованого дизерингу (таблиця 3.3, рисунок 3.23).

Таблиця 3.3 – Загальний час роботи (в секундах) модуля з алгоритмами програмного забезпечення для детермінованого упорядкованого дизерингу

	Алгоритми упорядкованого дизерингу									
Тестове зображення	Алгоритм розсіювання помилок Флойда–...	Алгоритм Джарвіса–Джудіса–...	Алгоритм Шітер Штуки	Алгоритм Аткінсона	Алгоритм Беркеса	Алгоритм Сієрри	Алгоритм Байєра	Алгоритм Беркеса	Алгоритм Сієрри	Алгоритм Байєра
1	0,084	0,030	0,110	0,026	0,135	0,156	0,017	0,125	0,015	0,010
2	0,205	0,233	0,112	0,044	0,238	0,209	0,131	0,166	0,031	0,024
3	0,215	0,258	0,167	0,088	0,321	0,238	0,210	0,219	0,121	0,045
4	0,323	0,484	0,402	0,330	0,390	0,271	0,215	0,245	0,286	0,150
5	0,326	0,503	0,479	0,332	0,480	0,342	0,220	0,277	0,371	0,233
6	0,467	0,570	0,527	0,378	0,698	0,364	0,225	0,319	0,499	0,259
7	0,468	0,607	0,579	0,578	0,883	0,474	0,413	0,374	0,593	0,485
8	0,530	0,733	0,652	0,733	0,907	0,489	0,750	0,537	0,944	0,519
9	0,891	0,779	0,933	0,768	1,985	0,670	0,774	0,599	1,674	0,672
10	0,987	1,306	1,116	1,857	2,256	0,899	1,221	0,715	1,699	0,681
11	1,066	1,836	1,728	1,970	2,284	0,915	2,015	1,044	2,232	0,915
12	1,785	2,889	2,325	3,031	2,396	1,359	2,379	1,624	2,388	2,146
13	2,153	3,110	2,648	3,041	2,541	1,360	2,834	2,202	3,070	2,673
14	2,693	3,748	3,432	4,269	2,791	2,732	3,179	2,668	3,609	3,198
15	2,895	5,244	3,775	5,642	2,946	3,498	4,078	4,162	3,737	4,701
16	3,045	6,789	3,957	6,457	3,810	3,506	4,153	4,421	4,060	5,608
17	4,319	7,095	4,217	6,892	6,931	5,206	4,506	4,853	5,646	6,896
18	5,624	7,242	6,838	7,181	8,298	5,641	4,519	5,750	5,681	7,655
19	9,162	10,863	7,838	7,820	10,925	8,043	6,518	6,407	6,241	9,382
20	19,652	16,254	9,443	35,302	18,475	33,697	11,399	12,145	16,755	30,349

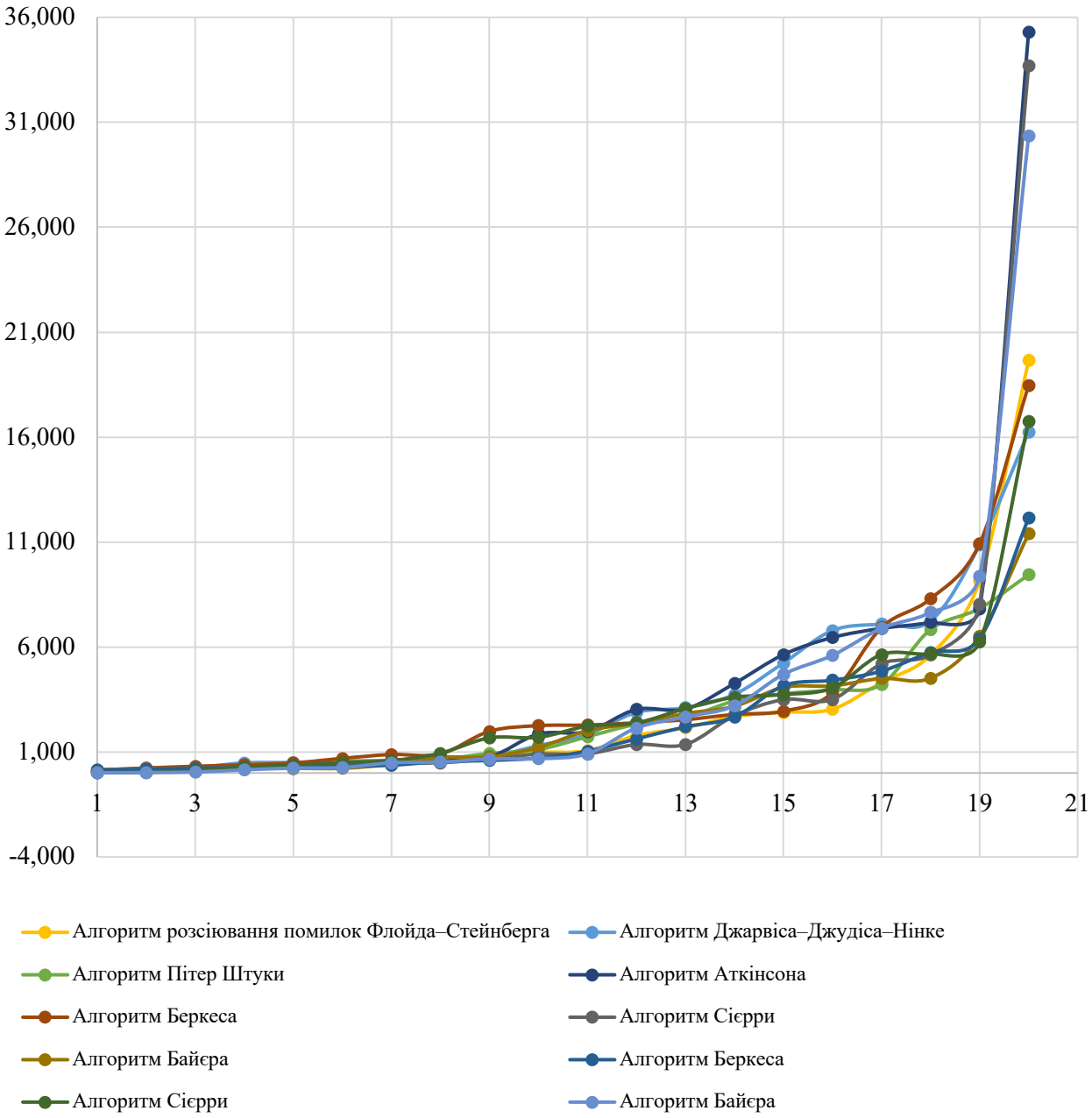


Рисунок 3.23 – Загальний час роботи (в секундах) модуля з алгоритмом програмного забезпечення для детермінованого упорядкованого дизерингу

Нижче наведені результати тестування методів випадкового дизерингу (таблиця 3.4, рисунок 3.24).

Таблиця 3.4 – Загальний час роботи (в секундах) модуля з алгоритмами програмного забезпечення для випадкового дизерингу

	Алгоритми випадкового дизерингу			
Тестове зображення	Простий випадковий дизеринг	Рівномірний випадковий дизеринг	Гауссовий випадковий дизеринг	Випадковий упорядкований дизеринг
1	0,056	0,013	0,058	0,035
2	0,081	0,126	0,082	0,114
3	0,097	0,151	0,176	0,130
4	0,140	0,151	0,216	0,184
5	0,182	0,157	0,228	0,191
6	0,287	0,165	0,255	0,324
7	0,551	0,189	0,279	0,518
8	0,561	0,224	0,453	0,617
9	1,095	0,318	1,012	0,659
10	1,154	0,420	1,156	0,723
11	1,185	0,424	1,206	0,962
12	1,224	0,511	1,991	1,429
13	1,309	1,272	2,177	1,736
14	2,213	1,350	2,239	2,536
15	2,987	1,351	2,255	2,667
16	3,604	1,793	2,380	2,752
17	4,916	2,447	4,338	4,738
18	5,053	2,564	4,757	4,999
19	5,589	2,987	5,779	5,408
20	7,687	14,445	24,478	19,129

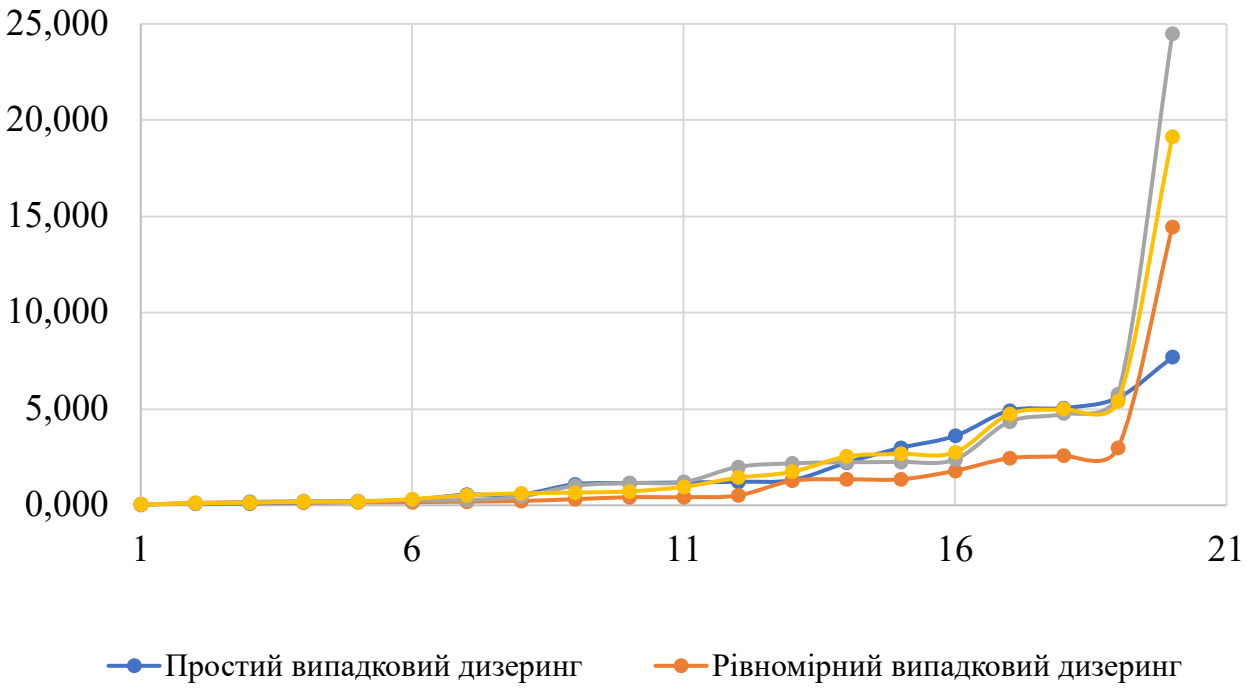


Рисунок 3.24 – Загальний час роботи (в секундах) модуля з алгоритмами програмного забезпечення для випадкового дизерингу

Детерміновані методи, такі як дизеринг Флойда-Стейнберга, забезпечують високу якість зображення, але є схильними до появи різних артефактів. Випадкові методи загалом менш схильні до регулярних візуальних артефактів, але мають трохи нижчу якість зображення, яка була обчислена на основі PSNR.

Пам’ять кожного з алгоритмів обмежується лише пам’яттю зберігання зображення, тому її не є доцільним наводити.

ВИСНОВКИ

У кваліфікаційній роботі було проаналізовано й програмно реалізовано різні алгоритми дизерингу, включаючи детерміновані й випадкові підходи. Було виявлено, що кожен метод має свої переваги та недоліки, залежно від контексту використання та вимог до якості зображення. Слід зазначити, що детерміновані методи, такі як дизеринг Флойда-Стейнберга, забезпечують високу якість зображення, але є схильними до появи різних артефактів. Випадкові методи загалом менш схильні до регулярних візуальних артефактів, але мають трохи нижчу якість зображення, яка була обчислена на основі PSNR. Також слід зазначити, що випадкові методи дизерингу показали хороші результати в порівнянні з детермінованими методами, особливо в контексті зменшення візуальних артефактів і збереження природності зображення.

У роботі також було продемонстровано, що використання випадкових методів може бути ефективним для різноманітних типів зображень, включаючи фотографії та різні технічні зображення, які формуються з різним освітленням.

Усі алгоритми дизерингу (випадкового і детермінованого) реалізовані з використанням мови програмування (інтерпретатора) Python. Було практично доведено, що випадкові методи дизерингу можуть бути ефективними для покращення якості зображень в умовах обмежених ресурсів.

Для більш детального аналізу алгоритмів дизерингу можна провести додаткові дослідження для оцінки ефективності алгоритмів на різних типах зображень та в різних умовах. Це може включати дослідження впливу різних параметрів алгоритмів на якість зображень.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Aach T., Mota C., Barth E. Analysis of halftoning and inverse halftoning using a linear signal model // IEEE Transactions on Image Processing. 2006. Vol. 15, No. 8. P. 2445-2459. DOI: https://doi.org/10.1109/TIP.2006.877529.
2. Allebach J. P., Pappas T. N. Digital halftoning // Introduction to inverse problems in imaging / Eds. M. Bertero, P. Boccacci. CRC Press, 2008. P. 181-212.
3. Cheng L., Bouzerdoum A., Phung S. L. A novel approach to document image binarization using fuzzy logic and clustering // IEEE Transactions on Systems, Man, and Cybernetics: Systems. 2014. Vol. 44, No. 4. P. 500-512. DOI: https://doi.org/10.1109/TSMC.2013.2283309.
4. Durand F. A frequency analysis of light transport // ACM Transactions on Graphics. 2015. Vol. 34, No. 4. P. 117:1-117:14. DOI: https://doi.org/10.1145/2766994.
5. Esfahani P. M., Asadi H., Jalali A. A new dithering algorithm for pseudo-color images based on optimized thresholds // IEEE Transactions on Image Processing. 2019. Vol. 28, No. 12. P. 6013-6026. DOI: https://doi.org/10.1109/TIP.2019.2923762.
6. Kang H. R. Computational color technology. SPIE Press, 2009.
7. Knuth D. E. Digital halftoning by means of mathematical morphology // Journal of Imaging Science and Technology. 2020. Vol. 64, No. 5. P. 1-15. DOI: https://doi.org/10.2352/J.ImagingSci.Technol.2020.64.5.050403.
8. Liao Q., Manduchi R. Efficient halftoning using deep learning // IEEE Transactions on Image Processing. 2021. Vol. 30. P. 3793-3804. DOI: https://doi.org/10.1109/TIP.2021.3059256.
9. Liang H., Zhang X. Improved error diffusion dithering method for high dynamic range images // Optics Express. 2017. Vol. 25, No. 19. P. 22858-22872. DOI: https://doi.org/10.1364/OE.25.022858.
10. Lipp M., Bruckner S., Gröller M. E. Fast and reliable error diffusion for multi-level halftoning // IEEE Transactions on Visualization and Computer Graphics. 2016. Vol. 22, No. 1. P. 1053-1062. DOI: https://doi.org/10.1109/TVCG.2015.2468593.

11. Ostromoukhov V., Hersch R. D. Digital halftoning for color and grayscale images // Academic Press Library in Signal Processing / Eds. R. Chellappa, S. Theodoridis. Academic Press, 2018. Vol. 5. P. 259-296.
12. Reeves T., McCann J. A practical approach to digital halftoning for modern displays // Journal of the Society for Information Display. 2017. Vol. 25, No. 7. P. 383-392. DOI: https://doi.org/10.1002/jsid.595.
13. Zhou Y., Chen T. Halftone image watermarking: A survey // ACM Computing Surveys. 2015. Vol. 48, No. 4. Article 52. DOI: https://doi.org/10.1145/2818332.
14. Esfahani P. M., Asadi H., Jalali A. A new dithering algorithm for pseudo-color images based on optimized thresholds // IEEE Transactions on Image Processing. 2021. Vol. 30. P. 6013-6026. DOI: https://doi.org/10.1109/TIP.2021.2923762.
15. Liao Q., Manduchi R. Efficient halftoning using deep learning // IEEE Transactions on Image Processing. 2021. Vol. 30. P. 3793-3804. DOI: https://doi.org/10.1109/TIP.2021.3059256.
16. Liang, H., & Zhang, X. (2022). Improved error diffusion dithering method for high dynamic range images. Optics Express, 30(1), 284-299. https://doi.org/10.1364/OE.441497.
17. Lipp M., Bruckner S., Gröller M. E. Fast and reliable error diffusion for multi-level halftoning // IEEE Transactions on Visualization and Computer Graphics. 2022. Vol. 28, No. 1. P. 1053-1062. DOI: https://doi.org/10.1109/TVCG.2021.3061236
18. Ostromoukhov V., Hersch R. D. Digital halftoning for color and grayscale images // Academic Press Library in Signal Processing / Eds. R. Chellappa, S. Theodoridis. Academic Press, 2021. Vol. 5. P. 259-296.
19. Reeves T., McCann J. A practical approach to digital halftoning for modern displays // Journal of the Society for Information Display. 2021. Vol. 29, No. 7. P. 383-392. DOI: https://doi.org/10.1002/jsid.595.
20. Zhou Y., Chen T. Halftone image watermarking: A survey // ACM Computing Surveys. 2021. Vol. 53, No. 4. Article 52. DOI: https://doi.org/10.1145/343981.