

# ТИТУЛЬНИЙ АРКУШ

## **ЗАВДАННЯ**

## **АНОТАЦІЯ**

У даній роботі розглянуто проблему усунення шуму на зображеннях. Досліджено існуючі методи усунення шуму, такі як Non-Local Means, Discrete Wavelet Transform, Ridgelet Transform. Проведено дослідження програмного забезпечення для усунення шуму для мобільних та настільних платформ.

Запропоновано модифікований метод Ridgelet-перетворення для усунення шуму на зображенні, що реалізується на графічному процесорі. Були проведені дослідження розробленого методу, що дозволили оцінити його ефективність та швидкодію у порівнянні з існуючими методами.

За результатами магістерської кваліфікаційної роботи подано до друку статтю «Investigation of Existing Image Denoising Algorithms», O. Pavliuk, R. Kutelmakh у Вісник Національного університету «Львівська політехніка».

## **ANNOTATION**

The problem of noise removal in images was considered in this paper. The existing noise removal methods such as the Non-Local Means, Discrete Wavelet Transform, Ridglet Transform were analysed. Mobile and desktop software for noise removal was explored.

A modified Ridgelet Tranform algorithm was proposed to eliminate noise in the image, which is implemented on GPU. It was developed by the research method, to provide for its effectiveness and performance compared to existing methods.

As a result of the master's qualification work is the article «Investigation of Existing Image Denoising Algorithms», O. Pavliuk, R. Kutelmakh in the Visnyk of Lviv Polytechnic National University.

## ЗМІСТ

ВСТУП.....	8
РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ СУЧASNIX МЕТОДІВ УСУНЕННЯ ШУMU НA ЗОБРАЖЕННЯХ.....	10
1.1 Проблема шуму в цифрових зображеннях.....	10
1.2. Огляд існуючих методів усунення шуму на зображеннях.....	13
1.3. Огляд існуючого ПЗ для усунення шуму на зображеннях .....	17
1.4. Формулювання завдання магістерського дослідження.....	21
РОЗДІЛ 2. АНАЛІЗ АЛГОРІТМІВ УСУНЕННЯ ШУMU НA ЗОБРАЖЕННЯХ ТА ОБГРУНТУВАННЯ ПЕРЕВАГ ЇХ ВИКОРИСТАННЯ.....	23
2.1. Опис запропонованого методу.....	23
2.2. Методологія проведення досліджень.....	25
2.3. Порівняльний аналіз алгоритмів для усунення шуму на зображеннях.....	26
2.4. Опис математичного апарату вирішення проблеми.....	26
2.5. Оцінка обчислювальної складності.....	28
2.6. Підтримка векторних математичних функцій мовою GLSL.....	29
РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ УСУНЕННЯ ШУMU НA ЗОБРАЖЕННЯХ.....	29
3.1. Специфікація вимог до програмного засобу.....	29
3.1.1. Призначення, мета.....	29
3.1.2. Перспективи продукту.....	30
3.1.3. Характеристики продукту.....	30
3.1.4. Класи користувачів та їх характеристики.....	30
3.1.5. Середовище функціонування.....	30
3.1.6. Обмеження проектування і реалізації.....	32
3.1.7. Документація користувача.....	32
3.1.8. Характеристики системи.....	32
3.1.9. Вимоги зовнішніх інтерфейсів.....	33
3.1.9.1. Користувацькі інтерфейси.....	33
3.1.9.2. Програмні інтерфейси.....	33
3.1.10. Вимоги продуктивності.....	33
3.2. План верифікації та валідації.....	34
3.2.1. Мета.....	34
3.2.2. Межі документу .....	34
3.2.3. Стратегія тестування компонент та процедур.....	34

3.2.4. План тестування.....	35
3.3. Звіт про тестування.....	35
3.4. Інструкція користувача.....	36
3.5. Обґрунтування технологій програмної реалізації.....	36
3.6. План забезпечення якості програмного продукту.....	37
3.6.1. Мета.....	37
3.6.2. Управління.....	38
3.6.2.1. Організація.....	38
3.6.2.2. Завдання.....	38
3.6.2.3. Ролі та обов'язки.....	39
3.6.2.4. Оцінка ресурсів, необхідних для забезпечення якості.....	39
3.6.3. Документація.....	39
3.6.3.1. Мета.....	39
3.6.3.2. Мінімальні вимоги до документації.....	40
3.6.3.2.1. Опис вимог до програмного забезпечення (SRD).....	40
3.6.3.2.2. Опис дизайну програмного забезпечення (SDD).....	40
3.6.3.2.3. Плани верифікації та валідації.....	41
3.6.3.2.4. Звіт з верифікації та валідації.....	41
3.6.3.2.5. Користувацька документація .....	41
3.6.3.2.6. План конфігураційного управління (SCMP).....	41
3.6.3.3. Інші документи.....	41
3.6.4. Стандарти, практики, конвенцій, і метрики.....	41
3.6.5. Тестування.....	42
3.6.6. Звітність проблем і коригувальних дій.....	42
3.6.7. Засоби, методи і методики .....	43
3.6.8. Контроль середовища.....	43
3.6.9. Контроль розробника .....	43
3.6.10. Збір коментарів, їх підтримка і збереження.....	43
3.6.11. Навчання.....	43
3.6.12. Управління ризиками.....	44
3.7. Опис схеми реалізації алгоритму FFT на GPU.....	44
3.8. Опис схеми реалізації алгоритму перетворення Радона на GPU.....	47
<b>РОЗДІЛ 4. ДОСЛІДЖЕННЯ АЛГОРИТМУ SHADERRIDGELET ДЛЯ ЗАДАЧІ УСУНЕННЯ ШУМУ НА ЗОБРАЖЕННЯХ.....</b>	<b>49</b>
4.1. Дослідження результатів перетворення Радона.....	49

4.2. Дослідження результатів перетворення Радона.....	49
4.3. Дослідження візуальної якості запропонованого методу.....	51
4.4. Дослідження часу роботи запропонованого методу для усунення шуму ShaderRidgelet .....	55
<b>РОЗДІЛ 5. ЕКОНОМІЧНА ХАРАКТЕРИСТИКА ПРОГРАМНОЇ СИСТЕМИ АЛГОРИТМУ УСУНЕННЯ ШУМУ НА ЗОБРАЖЕННЯХ.....</b>	<b>57</b>
5.3. Визначення комплексного показника якості.....	65
5.4. Визначення експлуатаційних витрат.....	67
5.5 Розрахунок ціни споживання проектного рішення.....	71
5.6. Визначення показників економічної ефективності.....	73
<b>ВИСНОВКИ.....</b>	<b>77</b>
<b>СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ.....</b>	<b>78</b>
<b>ДОДАТОК А. Результати роботи алгоритму для зображень зі значенням шуму sigma=0.18..80</b>	<b>80</b>
<b>ДОДАТОК Б. Результати роботи алгоритму для зображень зі значенням шуму sigma=0.18..81</b>	<b>81</b>
<b>ДОДАТОК В. Програмний код алгоритму DWT.....</b>	<b>82</b>

## **ПЕРЕЛІК СКОРОЧЕНЬ І СИМВОЛІВ**

ISO — світлоочутливість фотокамери

ПЗ — програмне забезпечення

ОС — операційна система

NLM — алгоритм нелокальних середніх

ДПФ — Дискретне перетворення Фур'є

FFT — Швидке перетворення Фур'є

CT — комп'ютерна томографія

GLSL — шейдерна мова OpenGL

GPU — графічний процесор

DWT — дискретне вейвлет-перетворення

D4 — вейвлет-коєфіцієнти Добеші

PCA — Метод головних компонент

mpss — швидкість роботи алгоритму, мегапіксель/сек

## ВСТУП

Обробка зображень — важлива сучасна галузь наукових досліджень. Задачі, що виникають у цій сфері, включають в себе як додавання певних ефектів з метою покращення зображення, так і усунення негативних явищ. Алгоритми поділяються на такі рівні:

- Г високорівневі алгоритми: розпізнавання та класифікація об'єктів тощо;
- Г алгоритми середнього рівня: будують моделі зображення, які потім можуть бути використані на високому рівні (знаходження країв).
- Г низькорівневі алгоритми: працюють з локальними характеристиками зображення: різноманітні фільтри (гаусівське розмиття для зображень з надто високим рівнем різкості, усунення шуму, додання контрасту).

Ефект шуму — досить поширена проблема для цифрових зображень. Вона має багато джерел походження, найпоширеніше з яких — це шум, отриманий в результаті нерівномірного потрапляння фотонів на сенсори фотокамери. Шум виникає в астрономічних зображеннях через велику віддаль спостережених об'єктів, комп'ютерній томографії через недостатню роздільну здатність сканувальних приладів, проте особливо часто з нею зустрічаються користувачі фотоапаратів з високою ISO та мобільних пристрій зі стандартними фотокамерами.

На даний час для усунення шуму на зображеннях розроблено багато алгоритмів та математичних моделей. Алгоритми для усунення шуму поділяються на два класи, залежно від того, в якій області працюють — просторовій чи частотній. Алгоритми, що працюють в просторовій області, враховують локальні відмінності зображення, в той час як алгоритми, що працюють в частотній області, розглядають частоти, що присутні в межах всього зображення.

Алгоритм, що забезпечує найкращу візуальну якість відновленого зображення на даний час – це Non-Local Means, або метод нелокальних середніх, розроблений французами Antoni Buades, Bartomeu Coll, та Jean-Michel Morel в 2011 році. Він розглядає фрагменти зображення – патчі, та встановлює нове значення пікселя як зважене середнє подібних патчів. Недоліком цього алгоритму є велика – квадратична – обчислювальна складність, що не підходить для роботи в режимі реального часу.

Основним підходом до усунення шуму в частотній області є вейвлети – функції з компактною підтримкою, що дозволяють розкласти сигнал на високі та низькі частоти. Простим прикладом вейвлета є вейвлет Хаара. Методи усунення шуму, що працюють в частотній області, застосовують до зображення один з видів вейвлет-перетворення, отримуючи набір коефіцієнтів та застосовуючи до них порогове відсікання. Вейвлети почали досліджуватись ще у 80-тих роках ХХ ст., проте більшість алгоритмів для усунення шуму на зображеннях були розроблені в період з 1999 до 2009 рр. До методів, що використовують переваги вейвлет-перетворення, належать алгоритми Ridgelet-перетворення та Curvelet-перетворення.

Алгоритми DWT та UDWT застосовують порогове відсікання коефіцієнтів, отриманих внаслідок застосування вейвлетів до рядків та стовпців зображення, в той час як Ridgelet-перетворення застосовує вейвлет-перетворення вздовж прямих зображень, використовуючи перетворення Фур'є.

У рамках магістерського дослідження розроблено новий метод для усунення шуму, який є модифікацією Ridgelet-перетворення та реалізований для виконання на графічному процесорі, що забезпечує високу швидкість роботи алгоритму.

## РОЗДІЛ 1. ОГЛЯД ТА АНАЛІЗ СУЧASНИХ МЕТОДІВ УСУНЕННЯ ШУМУ НА ЗОБРАЖЕННЯХ

### 1.1 Проблема шуму в цифрових зображеннях

Шум — це випадкові, відсутні на реальному зображенні варіації інтенсивності чи кольорової інформації на зображені. У загальному випадку шум на зображені може бути представлений як

$$v(i) = u(i) + n(i)$$

де  $i$  — це піксель зображення,  $v(i)$  — спостережене значення,  $u(i)$  — реальне значення  $i$ , яке обчислюється як середнє значення кількості фотонів, спостережених протягом тривалого періоду часу, і  $n(i)$  - це значення шуму.

Рівень шуму залежить від сигналу, таким чином чим більшим є “справжнє” значення  $u(i)$ , тим більшим є значення шуму  $n(i)$ . У існуючих моделях шуму, нормалізовані значення  $n(i)$  та  $n(j)$  для різних пікселів розглядаються як незалежні випадкові величини, в той час як Гаусівський шум — це білий(адитивний) шум.

Відомі алгоритми усунення шуму залежать від параметра фільтрування  $h$ . This parameter stands for filtering degree applied to the image. Для більшості алгоритмів цей параметр залежить від оцінки середнього квадратичного відхилення  $\sigma$  [1]. Таким чином, завданням алгоритмів усунення шуму є обчислення “справжнього” значення пікселя  $u(i)$ , беручи до уваги джерело зображення та параметр фільтрування  $h$ .

Типи шуму на зображеннях поділяють в залежності від джерела походження:

1) Гаусівський шум:

- | виникає внаслідок зчитування сенсорами кількості фотонів, викликаного поганим освітленням та/або високою температурою;
- | адитивний, незалежний для кожного пікселя;

Г у кольорових камерах, де більша вага надається синьому каналу, аніж зеленому чи червоному, більше шуму припадає саме на синій канал.

2) “Salt-and-pepper” (англ. Сіль-та-перець) шум

Г "імпульсний" шум;

Г темні пікселі в світлих частинах зображення і світлі пікселі в темних частинах зображення;

Г викликаний помилками аналогово-цифрового перетворювання, бітовими помилками транспортування тощо.

3) Шум фотокамери (Shot noise)

Г підлягає розподілу Пуассона, який, за винятком дуже низьких рівнів інтенсивності, апроксимує розподіл Гаусса;

Г викликаний статистичними коливаннями частинок, тобто, варіаціями в кількості фотонів на заданому рівні експозиції.

4) Шум квантизації (рівномірний шум)

Г викликаний квантуванням пікселів зчитуваного зображення на певну кількість рівнів (н-д, 255).

Г підлягає приблизно рівномірному розподілу.

5) Шум “зерниста плівка” (англ. “Film grain” [1])

Г має розподіл ймовірностей, близький до shot noise.

Г Якщо зерна плівки рівномірно розподілені (однакове число на одиницю площині), і кожне зерно має рівну та незалежно розподілену ймовірність затемнення після зчитування фотонів, тоді кількість таких темних зерен на зображенні буде випадковою та підлягатиме біноміальному розподілу.

Різні види шуму мають різне походження (погане освітлення, погана фотокамера, помилки конвертування, кадри фільму) і різні математичні моделі (Гаусівський розподіл, рівномірний розподіл, розподіл Пуассона).

Колірний простір YCrCb і його варіанти — один з популярних методів ефективного представлення кольорових зображень. Канал Y містить інформацію про освітленість. Показник може бути визначений як середнє зважене компонентів R, G, B, і записується у наступному вигляді:

$$Y = k_r * R + k_g * G + k_b * b ,$$

де коефіцієнти  $k$  — вплив кожного каналу на значення освітленості.

Інформація порівнянням може бути представлена як кольорова різниця (хроматичні дані чи насиченість кольору). У такому випадку кожна компонента обчислюється як різниця між R,G,B та освітленістю Y.

$$C_b = B - Y$$

$$C_r = R - Y$$

$$C_g = G - Y$$

Для повного опису кольорового зображення необхідно задати значення освітленості Y та хроматичних компонент Cb, Cr, Cg. Проте достатньо запам'ятати тільки дві хроматичні компоненти — третя завжди може бути обчислена. У колірній моделі YCrCb записується тільки значення яскравості Y та хроматичні компоненти Cr, Cb.

Важливою перевагою колірної моделі YCrCb над RGB є те, що компоненти Cr, Cb можуть бути представлені з нижчою роздільною здатністю, бо система людського зору більш чутлива до освітленості, ніж до кольору. Ця особливість ефективно використовується форматом JPEG для стиснення зображень.



**Рис. 1.1.** Зліва направо: канали Y, Cr, Cb

Тому алгоритми усунення шуму працюють з кожним каналом окремо, таким чином, отримуючи на вхід одноканальне зображення. Як бачимо з вищезгаданого зображення, більше деталей зображення сконцентровано у каналі освітленості, і це необхідно врахувати при розробці ефективного алгоритму.

## 1.2. Огляд існуючих методів усунення шуму на зображеннях

Алгоритми, базовані на патчах

Патч — це прямокутна область зображення [2], що враховує локальні характеристики пікселя.



**Рис. 1.2.** Порівняння пікселів за допомогою патчів

Основна ідея алгоритмів, базованих на патчах — встановити нове, незашумлене значення пікселя як суму “зваженого середнього” (weighted average) з подібних патчів зображення, до якого застосували функцію згортки, наприклад, Гаусівський фільтр.

Найвідоміший алгоритм, базований на патчах — Non-Local Means[2]. Формулу для обчислення значення пікселя всередині патча записують у вигляді:

$$u_i(p) = \frac{1}{C(p)} \sum_{q \in B(p,r)} u_i(q) * w(p,q)$$

де  $i=1, 2, 3$  and  $B(p, r)$  позначає квадратну область “сусідство”, центром якої є піксель  $p$ , розміру  $2r+1 \times 2r+1$  пікселів. Ця зона пошуку обмежена вікном фіксованого розміру через обмеження на обчислювальну складність. Це вікно розміру  $21 \times 21$  для малих та середніх значень рівня шуму —  $\sigma$ . Для великих значень  $\sigma$  розмір вікна збільшують до  $35 \times 35$ , для того, щоб знайти більше подібних пікселів.

Вага  $w(p, q)$  залежить від квадратичної евклідової відстані  $d^2 = d^2(B(p,f), B(q,f))$  для  $2f+1 \times 2f+1$  кольорових патчів, центрами яких є  $p$  та  $q$  відповідно.

$$d^2(B(p,f), B(q,f)) = \frac{1}{3(2f+1)^2} \sum_{i=1}^3 \sum_{j \in B(0,f)} (u_i(p+j) - u_i(q+j))^2$$

Тобто, відновлене значення кожного пікселя обчислюється за допомогою значень сусідніх (за значеннями, а не за відстанню) пікселів. Для кожного пікселя нове значення буде зваженим середнім значень подібних пікселів.

Використовується експоненційне ядро для обчислення ваг  $w(p, q)$

$$w(p,q) = e^{-\frac{\max(d^2 - 2\sigma^2, 0.0)}{h^2}}$$

де  $\sigma$  позначає стандартну девіацію шуму, а  $h$  — параметр фільтрування, встановлений на основі значення  $\sigma$ . Функція ваги встановлюється таким чином, щоб усереднити значення подібних патчів. Тобто, вага патчів з квадратичною відстанню, меншою, ніж  $2\sigma^2$  встановлюється в 1, в той час як вага патчів з більшою відстанню зменшується експоненційно.

Обчислювальна складність NLM —  $O(n^2 n^2 w^2 p^2)$ , де  $n$  — розмір зображення,  $w$  — розмір вікна пошуку,  $p$  — розмір патча. Така складність близька до квадратичної, що неефективно для зображень великого розміру.

Існує багато модифікацій NLM. Один з підходів [3,4] використовує PCA для визначення вектора основних характеристик патча, таким чином, суттєво зменшуючи складність обчислення відстані між патчами, але жертвуючи достовірністю такої відстані.

Ефективний алгоритм, який покращує Non-Local Means — це метод, що будує кластерне дерево для простору патчів, таким чином логарифмічно зменшуючи обчислювальну складність пошуку подібних патчів [4]. Для поділу патчів застосовується метод кластеризації k-means.

Обчислювальна складність алгоритму належить до класу  $O(n * \log n)$ , де  $n$  — розмір зображення.

### Алгоритми, базовані на вейвлетах

Ідея цифрового вейвлет-перетворення полягає у тому, що сигнал ділиться на дві смуги з високою та низькою частотами. Шум залишається у високій частоті, до нього застосовують порогове відсікання.

Вейвлети — це функції, для яких виконуються певні математичні умови. Назва “вейвлет” передбачає, що сума функції на всій області визначення повинна дорівнювати нулю, “коливаючись” навколо х-осі. Функція вейвлета повинна бути добре локалізована. Інші вимоги є технічними і вимагаються для забезпечення швидкого та точного обчислення прямого та зворотнього вейвлет-перетворень.

Впродовж трьох останніх десятиліть розроблено багато типів вейвлетів: гладкі вейвлети, вейвлети з компактною підтримкою, вейвлети з простим математичним виразом тощо. Функція вейвлета задається формулою

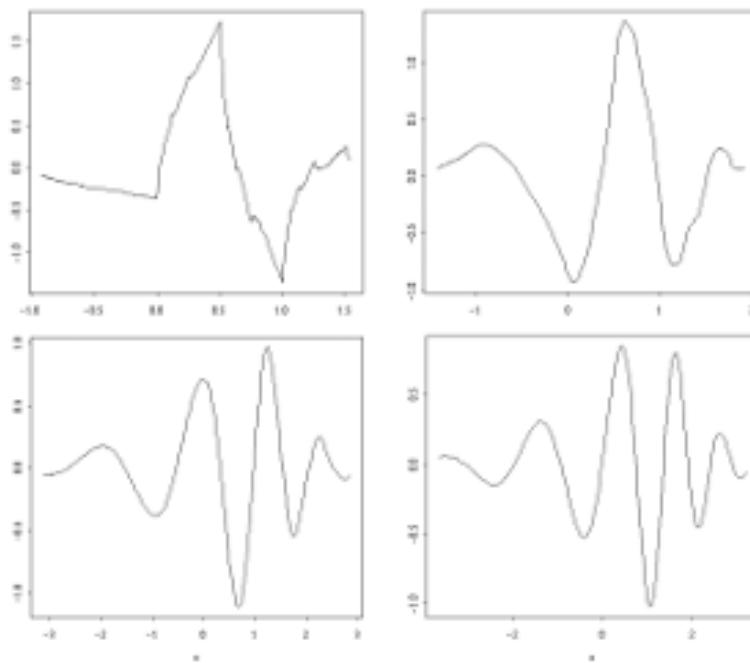
$$\psi_{s,u}(t) = \frac{1}{\sqrt(s)} \psi \frac{(t-u)}{s}$$

де  $s$  — це масштаб, а  $u$  — зсув у часі.

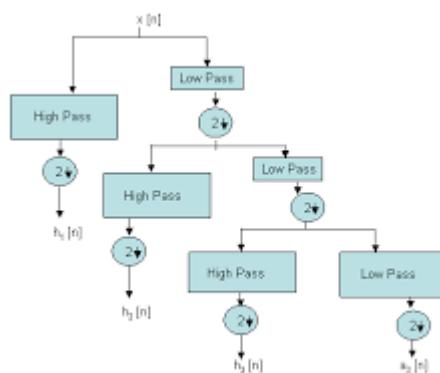
Неперервне вейвлет-перетворення функції  $f(t)$  обчислюється як згортка вхідного сигналу з функцією вейвлета.

$$Wf(s, u) = \int_{-\infty}^{\infty} f(t) \psi_{s,u}(t) dt$$

Найвидомішими є вейвлет Хаара та вейвлет Добеші.

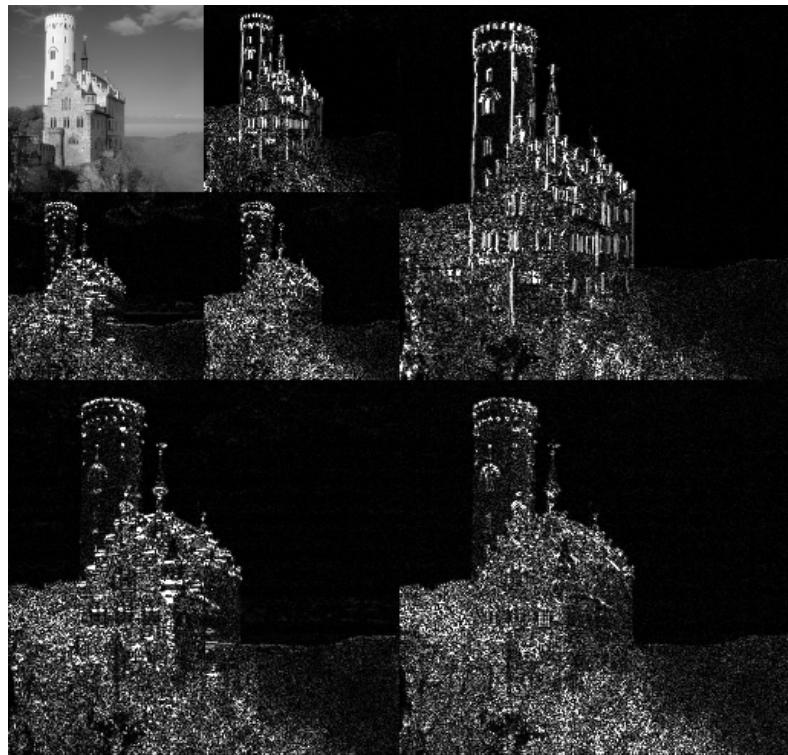


**Рис. 1.3** Вейвлет Добеші



**Рис. 1.4** Декомпозиція сигналу у дискретному вейвлет-перетворенні

Дискретне вейвлет-перетворення для цифрового зображення відрізняється від одновимірного дискретного DWT тільки тим, що спочатку застосовують вейвлет-функцію до рядків зображення, а потім до стовпців.



**Рис. 1.5.** Дискретне вейвлетне перетворення, яке застосовується у JPEG2000

### 1.3. Огляд існуючого ПЗ для усунення шуму на зображеннях

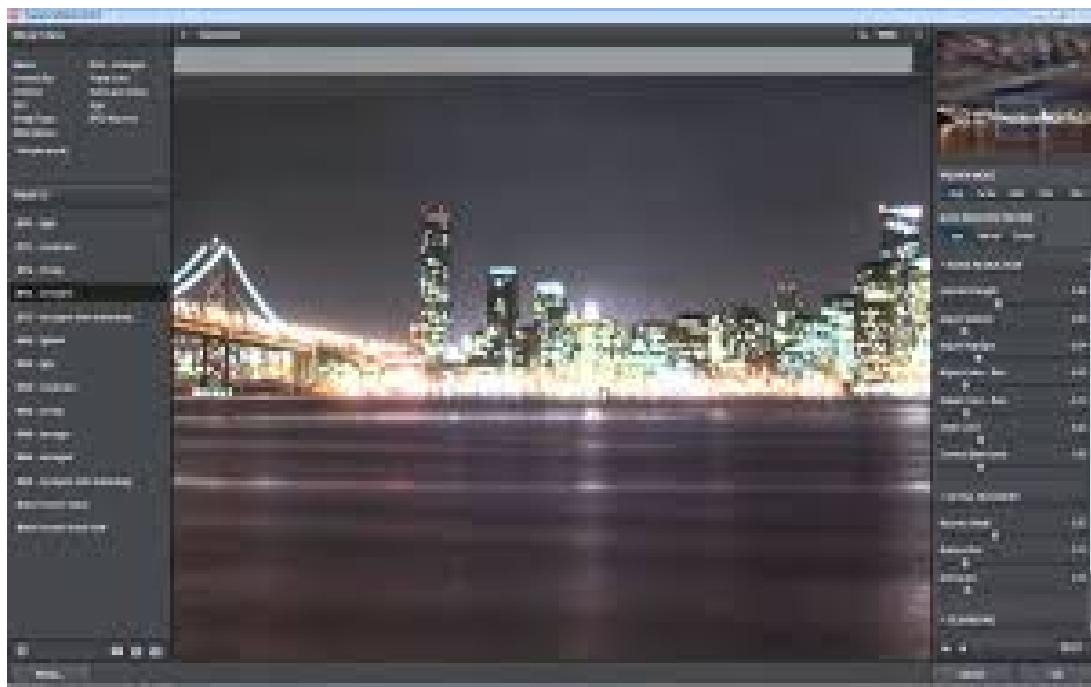
На ринку настільних та мобільних платформ представлено багато програмного забезпечення для усунення шуму на зображеннях, тому доцільно дослідити якість роботи та особливості інтерфейсу користувача популярних програм.

ПЗ для усунення шуму для настільних платформ поділяється на такі типи: окремі (standalone) програми та плагіни до таких відомих редакторів зображень як LightRoom та Aperture.

ПЗ для усунення шуму для мобільних платформ має значно нижчу швидкість роботи через обмежені ресурси процесора, за умови такого ж якісного алгоритму та зручного інтерфейсу користувача.

#### Topaz Denoise (Plugin)

Це плагін для таких програм обробки зображень як Lightroom, Aperture.



**Рис. 1.6.** Програма Topaz Denoise

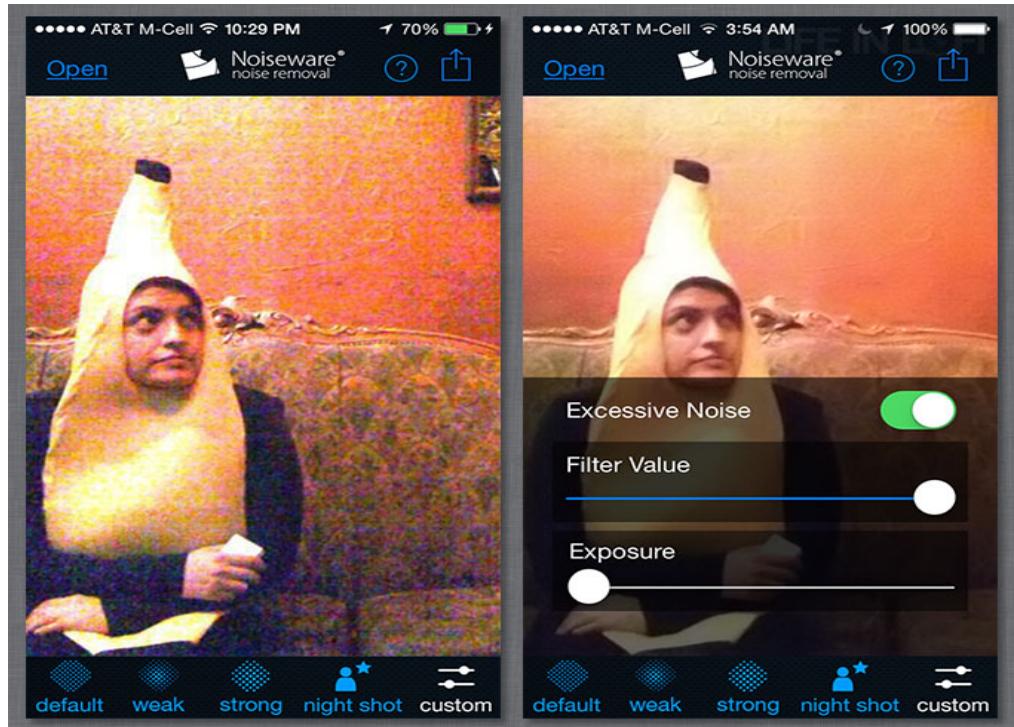
Тут є налаштування (Noise Reduction, Detail Recovery, Debanding) та 12 пресетів, які визначають значення тих налаштувань. Пресети враховують тип камери (DSLR / Point and Shoot), тип зображення (JPEG / RAW), ISO (low / median / high / very high). Тобто, пресет може вибрати не тільки користувач, але й програма на основі даних про файл.

Сам інтерфейс пресетів надто технічний, тому що користувач може не знати, яка в нього ISO, камера тощо, зате він зрозуміліший, ніж “Soft” чи “Balanced” (як у Noiseless pro). У Noise Reduction є Adjust Shadow, Adjust Highlight, Correct Black Level etc. У Preview Mode можна вибрати тільки один канал якоїсь кольорової моделі (R, B, H, Grayscale). Topaz Denoise враховує важливу проблему втрати деталей: для цього в налаштуваннях є вкладка Detail Recovery (reduce blur, add grain).

### NoiseWare (Plugin)

Award-winning plugin. Тут є пресети, але на відміну від Topaz Denoise, вони враховують не характеристики камери, а місце зйомки (Night shots, People photographs, Film grain) та рівень шуму(Weaker noise, Full noise, Full stronger noise etc).

## NoiseWare (Mobile)



**Рис. 1.7.** Програма NoiseWare

Як і в настільній версії, тут є пресети, але їх менше. I custom mode має тільки два контроли, але в мобільній версії їх і не може бути багато.

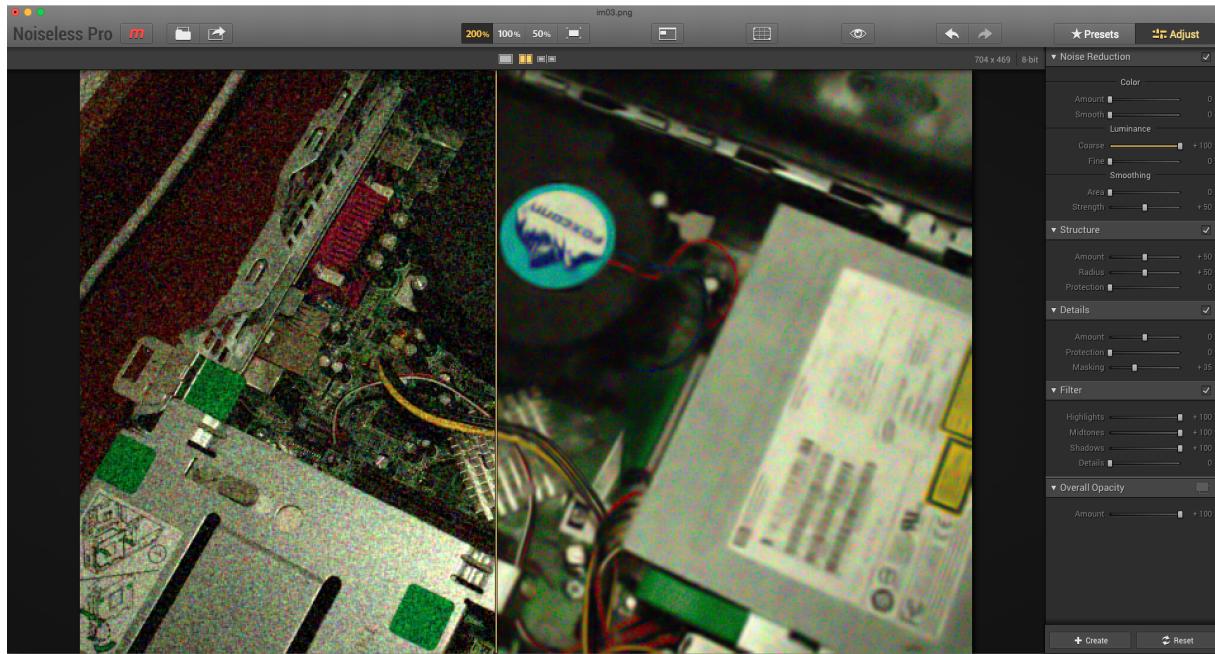
## Noiseless Pro (macphun) (Plugin)

Теж є налаштування для Detail: Enhancement & Protection. У Noise Reduction є параметри Luminance & Color (ці параметри інтуїтивно зрозумілі – шум є окремо в кольорі, окремо в освітленості, і їх потрібно контролювати окремо).

Ще є Tonal & Color range (RGB+CMYK каналі – ступінь впливу алгоритму на кожен з каналів) і Frequency (Noise Level і Noise Reduction).

Виглядає, що це найкраща denoising програма (висока якість усунення шуму та логіка налаштувань).

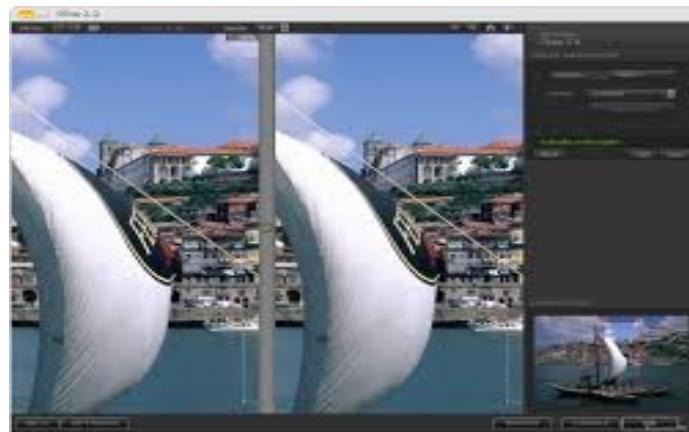
Original/fixed розділені вертикальною лінією (але є і традиційні view modes).



**Рис. 1.8.** Програма Noiseless Pro

Topaz Denoise, Noiseware etc.

Dfine (Nick Collection) (Plugin)



**Рис. 1.9.** Програма Dfine

Dfine дуже відрізняється від інших програм:

“–” немає пресетів (*Automatic profile applied*).

“–” немає налаштувань для ***Detail protection***.

“+” є режим ***Control points***.

## Neat Image Demo-Version (Standalone)

standalone

NoiseWare      Topaz Denoise.

Output image, Device Noise Profile (

, ).      Filter Settings

Color, Sharpening Amount.

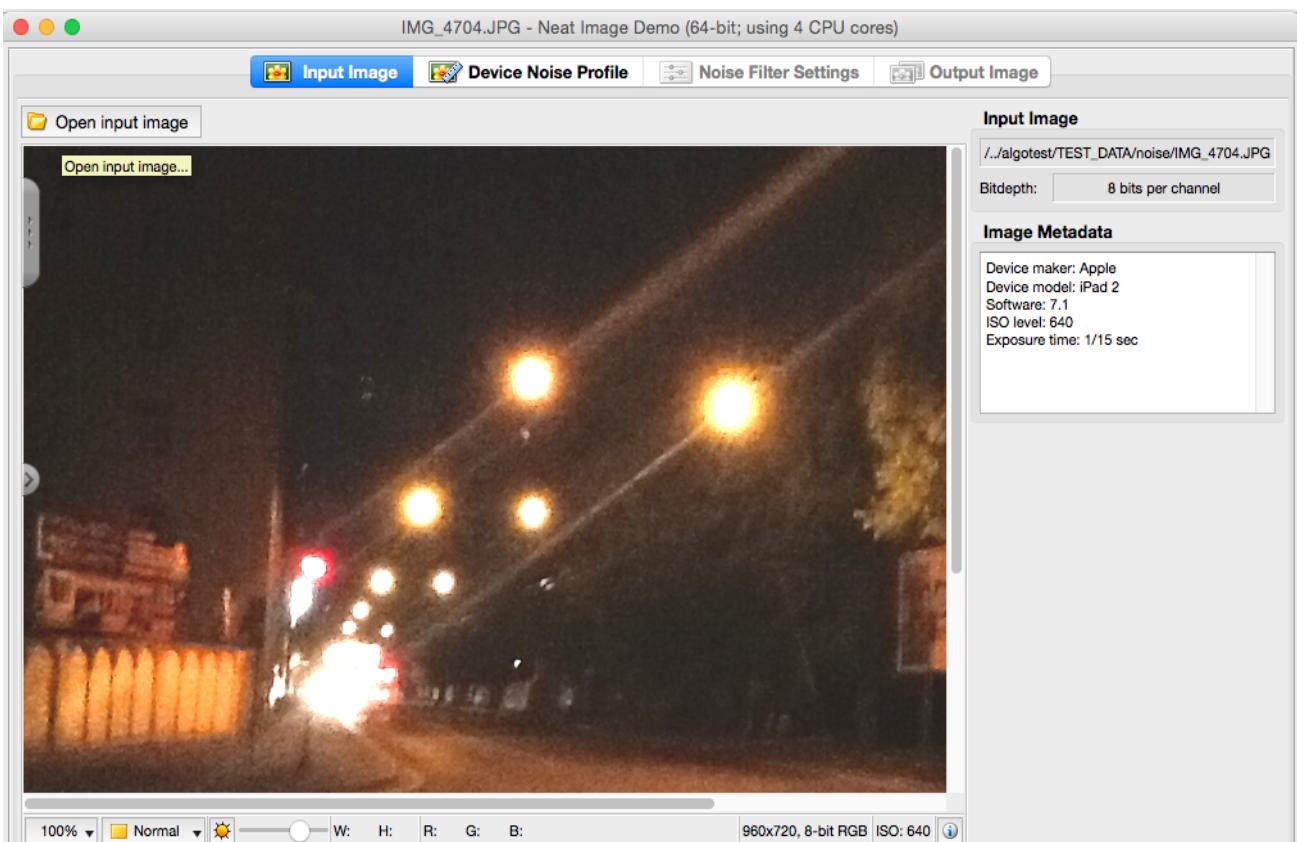


Рис. 1.10. Програма NeatImage

### 1.4. Формулювання завдання магістерського дослідження

Отже, **об'єктом** дослідження у даній магістерській роботі буде процес усунення шуму на зображеннях.

**Предметом** дослідження будуть методи, алгоритми та програмне забезпечення для усунення шуму на зображеннях.

**Метою** цієї роботи є розроблення алгоритмів та програмного забезпечення для усунення шуму на зображеннях:

Поставлені такі **завдання**, які мають бути виконані в процесі роботи над дипломним проектом:

1. Дослідити існуючі методи усунення шуму на зображеннях
2. Розробити алгоритм усунення шуму, що враховує переваги існуючих методів та використовує можливості графічного процесора.
3. Розробити програму інтерфейсу користувача для платформи Mac OS X.
4. Розробити програму інтерфейсу користувача для платформи Android OS.

## РОЗДІЛ 2. АНАЛІЗ АЛГОРИТМІВ УСУНЕННЯ ШУМУ НА ЗОБРАЖЕННЯХ ТА ОБГРУНТУВАННЯ ПЕРЕВАГ ІХ ВИКОРИСТАННЯ

### 2.1. Опис запропонованого методу

У рамках магістерського дослідження було розроблено альтернативний метод усунення шуму на зображенні. Запропонований метод — це усунення шуму в коефіцієнтах Ridgelet-перетворення з покращеною схемою інтерполяції, розроблений для виконання на GPU. У алгоритмі Ridgelet-перетворення усунення шуму відбувається у просторі Радона, тобто, вздовж прямих — замість координат пікселів розглядаються координати прямих, інтенсивність для них означає суму пікселів на цій прямій. Формула перетворення Радона задається як функція від двох параметрів — кута та відстані до центру полярнох систем координат:

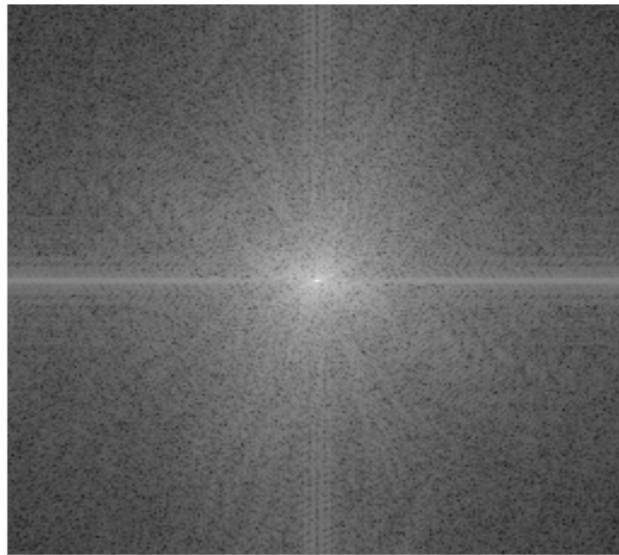
$$R_\mu f(\phi, s) = \int_{x \in L(\phi, s)} f(x) \mu(\phi, x) dx$$

Існує багато алгоритмів для отримання даного перетворення [link] з обчислювальною складністю близькою до квадратичної. Проте у частотній області перетворення Радона отримують за допомогою Projection-Slice Theorem.

Алгоритм Ridgelet-перетворення — це алгоритм, який працює в частотній області і використовує DWT. Він полягає в тому, що до зображення в перетворенні Радона застосовується дискретне вейвлет-перетворення з пороговим відсіканням коефіцієнтів. Кроки алгоритму полягають у наступному:

1. Зображення у форматі RGB перетворюється у формат YcrCb, для якого відбувається усунення шуму в кожному каналі.
2. Для обраного каналу зображення обчислюється FFT-перетворення (детальніше схема алгоритму FFT-перетворення описана в пп. 2.4 та 3.7),

центр координат FFT-перетворення встановлюється в середину зображення.

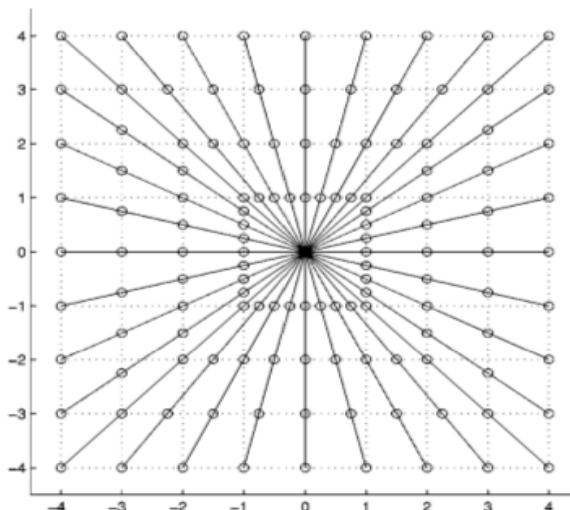


**Рис. 2.1** Приклад 2D FFT зображення

3. Для отриманого FFT зображення на основі теореми Projection-Slice обчислюється перетворення Радона. Для цього використовується ректополярна решітка (Рис. 2.2), яка є усередненням між полярною та декартовою системами координат. На цій решітці розглядаються промені, що проходять через центр системи координат.

Застосувавши до цих променів зворотнє перетворення Фур'є, отримують перетворення Радона.

Реалізація перетворення Радона для виконання на GPU наведена в п. 3.8.



**Рис. 2.2** Ректополярна решітка для зображення 8x8

4. До отриманого на попередньому кроці перетворення Радона застосовують вейвлет-перетворення з пороговим відсіканням коефіцієнтів, приклад якого наведено в п. 1.2. У якості вейвлета використовується вейвлет Добеші, значення порогу є параметром алгоритму і підбирається експериментально для кожного зображення залежно від рівня шуму. У проекті магістерського дослідження застосовано вейвлет Добеші  $D_4 = [0.482962, 0.836516, 0.224143, -0.129409]$ , висока та низька частота обчислюються за формулами:  $high[v] = y[2*v]*D_4[0] + y[2*v+1]*D_4[1] + y[2*v+2]*D_4[2] + y[2*v+3]*D_4[3]$

та

$$bw[v] = y[2*v]*D_4[3] - y[2*v+1]*D_4[2] + y[2*v+2]*D_4[1] - y[2*v+3]*D_4[0]$$

відповідно.

Вейвлет-коєфіцієнти з абсолютним значенням меншим за заданий поріг  $\sigma$  встановлюються в 0, потім застосовується обернене перетворення.

5. Після усунення шуму вздовж прямих у перетворенні Радона отримують нове FFT зображення, використовуючи ту ж схему інтерполяції: пікселям у точках на променях, які проходять через початок системи координат, встановлюють відповідне значення у рядку перетворення Радона.

У запропонованому методі використовується інша схема інтерполяції: замість встановлення значення найближчого сусіда нове значення у зображенні FFT обчислюється на основі чотирьох сусідів за формулою лінійної інтерполяції.

6. До отриманого зображення застосовується зворотне перетворення Фур'є.  
 7. Отримане зображення перетворюють назад у формат RGB.

## 2.2. Методологія проведення досліджень

Як уже згадувалося раніше, серед алгоритмів усунення шуму найвищу якість забезпечує алгоритм Non-Local Means, тому необхідно порівняти швидкість роботи та якість розробленого алгоритму і NLM на однакових

вхідних даних. Для цього розглянемо тестову базу зашумлених зображень, яка доступна на сайті алгоритму NLM [link].

### **2.3. Порівняльний аналіз алгоритмів для усунення шуму на зображеннях**

Оскільки NLM забезпечує найвищу якість з існуючих методів усунення шуму, то планується провести порівняльний аналіз візуальної якості отриманих результатів роботи розробленого алгоритму та алгоритму NLM для тестового набору зображень розміру 0.5 mp з середнім значенням шуму sigma=0.09, тип зображень — природа, об'єкти та архітектурні споруди.

### **2.4. Опис математичного апарату вирішення проблеми**

Перетворення Фур'є [link] – це базовий метод для всіх алгоритмів, що працюють з частотами. Як аналоговий, так і дискретний сигнал можна представити у вигляді суми синусоїд з різними амплітудами та зсувом. Ці синусоїди зберігаються у вигляді комплексних чисел  $a+ib$ , де  $r=\sqrt{a^2+b^2}$  – амплітуда синусоїди,  $\phi=\arctan(b)$  – кут радіус-вектора, або зсув фази синусоїди.

Дискретне перетворення Фур'є дозволяє отримати спектр частот сигналу. Пряме перетворення задається формулою:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot (\cos(-2\pi k \frac{n}{N}) + j \sin(-2\pi k \frac{n}{N})), \quad n \in \mathbb{Z}$$

Формула для зворотнього перетворення виглядає наступним так:

$$x_n = \frac{1}{N} \sum_{k=0}^{N-1} X_k \cdot (\cos(2\pi k \frac{n}{N}) + j \sin(2\pi k \frac{n}{N})), \quad n \in \mathbb{Z}$$

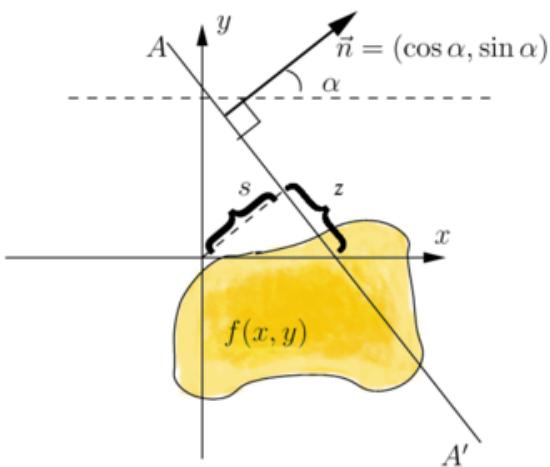
де  $N$  = довжина вхідного сигналу,  $x_n$  = значення сигналу в момент часу  $n$ ,  $X_k$  = кількість частоти  $k$  у сигналі (амплітуда та фаза).

Надзвичайно важливою особливістю даного перетворення є те, що операція згортки (convolution) сигналу з фільтром довільної довжини виконується за лінійний час. Для усунення шуму застосовують вейвлет-

фільтри, які теж можуть бути представлені у частотній області за допомогою комплексних вейвлетів.

Швидке перетворення Фур'є використовує властивості подання сигналу у Дискретному Перетворенні Фур'є (ДПФ), і зменшує обчислювальну складність алгоритму з квадратичної до лінійно-логарифмічної.

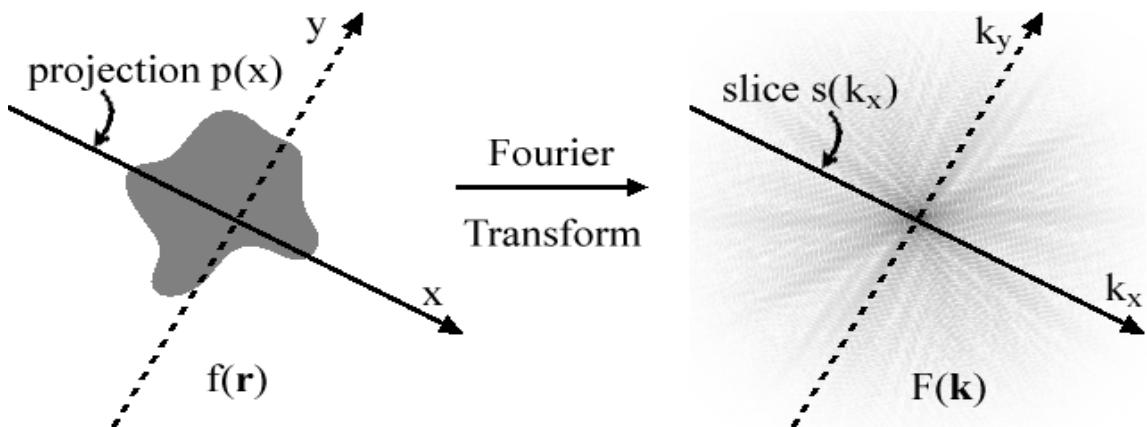
Перетворення Радона – це інтегральне перетворення, яке для кожної прямої на зображенні ставить їй у відповідність суму пікселів зображення на цій прямій.



**Рис. 2.3** Пряма як вектор довжини нормалі та кута у перетворенні Радона

Перетворення Радона широко використовується у томографії для відновлення зображень внутрішніх органів з рентгенівських знімків, коли промені проходять через об'єкт під різними кутами, таким чином накладаючись і спотворюючи проекцію.

Projection-Slice Theorem стверджує, що проекція функції на пряму, із застосованим до неї перетворенням Фур'є, рівна променю, що проходить через початок координат у області частот. А оскільки проекція функції на пряму – це сума значень перпендикулярних до цієї прямії ліній зображення, то кожна пряма на зображенні Радона рівна зворотньому перетворенню Фур'є променя (slice), що проходить через центр під відповідним кутом.



**Рис. 2.4** Зв'язок між перетворенням Фур'є та перетворенням Радона

$$\hat{f}(r\alpha) = \int_{-\infty}^{\infty} ds \int_{\mathbf{x} \cdot \alpha = s} e^{-2\pi i r(\mathbf{x} \cdot \alpha)} dm(\mathbf{x}).$$

Складність цього алгоритму лінійна, що дозволяє отримати перетворення Радона для декількох частин зображення, як описано в [5].

Ridgelet-перетворення – це вейвлет-перетворення, застосоване до перетворення Радона.

## 2.5. Оцінка обчислювальної складності

Обчислювальна складність розробленого алгоритму обчислюється на основі інформації про обчислювальну складність алгоритмів, які використовує ShaderRidgelet.

Зокрема, оскільки обчислювальна складність FFT становить  $O(n * \log(n))$ , складність перетворення Радона є лінійною, DWT з пороговим відсіканням виконується за час  $O(n * \log(n))$ , то складність запропонованого алгоритму ShaderRidgelet становить  $O(n * \log(n))$ . Оскільки алгоритм складається з кількох кроків, і кожен крок в нотації «O-велике» має свій лінійний коефіцієнт, то час роботи алгоритму буде залежати від суми цих коефіцієнтів.

Специфіка роботи шейдера передбачає векторне виконання операцій,

тому для алгоритму ShaderRidgelet лінійний коефіцієнт складності буде приблизно в 10 разів (стандартний множник) меншим, ніж у його процесорного аналога.

## **2.6. Підтримка векторних математичних функцій мовою GLSL**

Мова GLSL — OpenGL Shader Language — призначена для швидкого виконання однотипних операцій над великими обсягами даних за допомогою програм, що виконуються на графічному процесорі - шейдерів.

Для реалізації FFT та перетворення Радона корисними будуть такі типи даних:

float: числові значення з плаваючою точкою

vec2, vec3, vec4: вектори, які містять об'єкти типу float

ivec2, ivec3, ivec4: вектори, які містять об'єкти типу int

mat2, mat3, mat4: матриці розміру 2x2 3x3 і 4x4 відповідно, які містять об'єкти типу float

sampler2D: тип даних для роботи з текстурами.

# **РОЗДІЛ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ УСУНЕННЯ ШУМУ НА ЗОБРАЖЕННЯХ**

## **3.1. Специфікація вимог до програмного засобу**

### **3.1.1. Призначення, мета**

Розроблюваний програмний засіб є модифікованим Ridgelet-перетворенням для виконання на графічному процесорі. На основі реалізації алгоритму буде оцінено його ефективність у порівнянні з найкращим існуючим алгоритмом Non-Local Means.

### **3.1.2. Перспективи продукту**

Проведено дослідження програм-аналогів, з них найбільш конкурентно-спроможними є Topaz Denoise, NoiseWare та Dfine. Вони пропонують два типи програмних рішень – мобільні додатки та плагіни для популярних програм.

Програма, розроблена в рамках магістерського дослідження, поєднує швидкість та якість усунення шуму, тому зможе зайняти своє місце на ринку програмного забезпечення.

### **3.1.3. Характеристики продукту**

Програмне забезпечення для усунення шуму на зображені призначення для виконання користувачем таких дій:

1. Завантаження вхідного зображення.
2. Задання параметрів алгоритму.
3. Зміна параметрів алгоритму.
4. Отримання вихідного зображення.
5. Збереження вихідного зображення.

### **3.1.4. Класи користувачів та їх характеристики**

Дане програмне забезпечення орієнтоване на користувачів мобільних та настільних платформ Android OS та Mac OS X відповідно. Тому можна виділити такі класи користувачів:

- 1) Власники мобільних пристройів
- 2) Користувачі Mac OS X, які мають фотокамери з високою ISO.

### **3.1.5. Середовище функціонування**

Дослідження проводиться на комп'ютері з такими апаратними характеристиками:

Процесор 3,19 GHz Intel Core i5

Пам'ять 8 ГБ 1333 МГц DDR3

Стартовий диск MacOS

Графіка AMD Radeon HD 7xxx 1024 МБ

Розроблене ПЗ функціонуватиме на пристроях Android починаючи з версії 4.4.2 та Mac OS починаючи з версії 10.7.

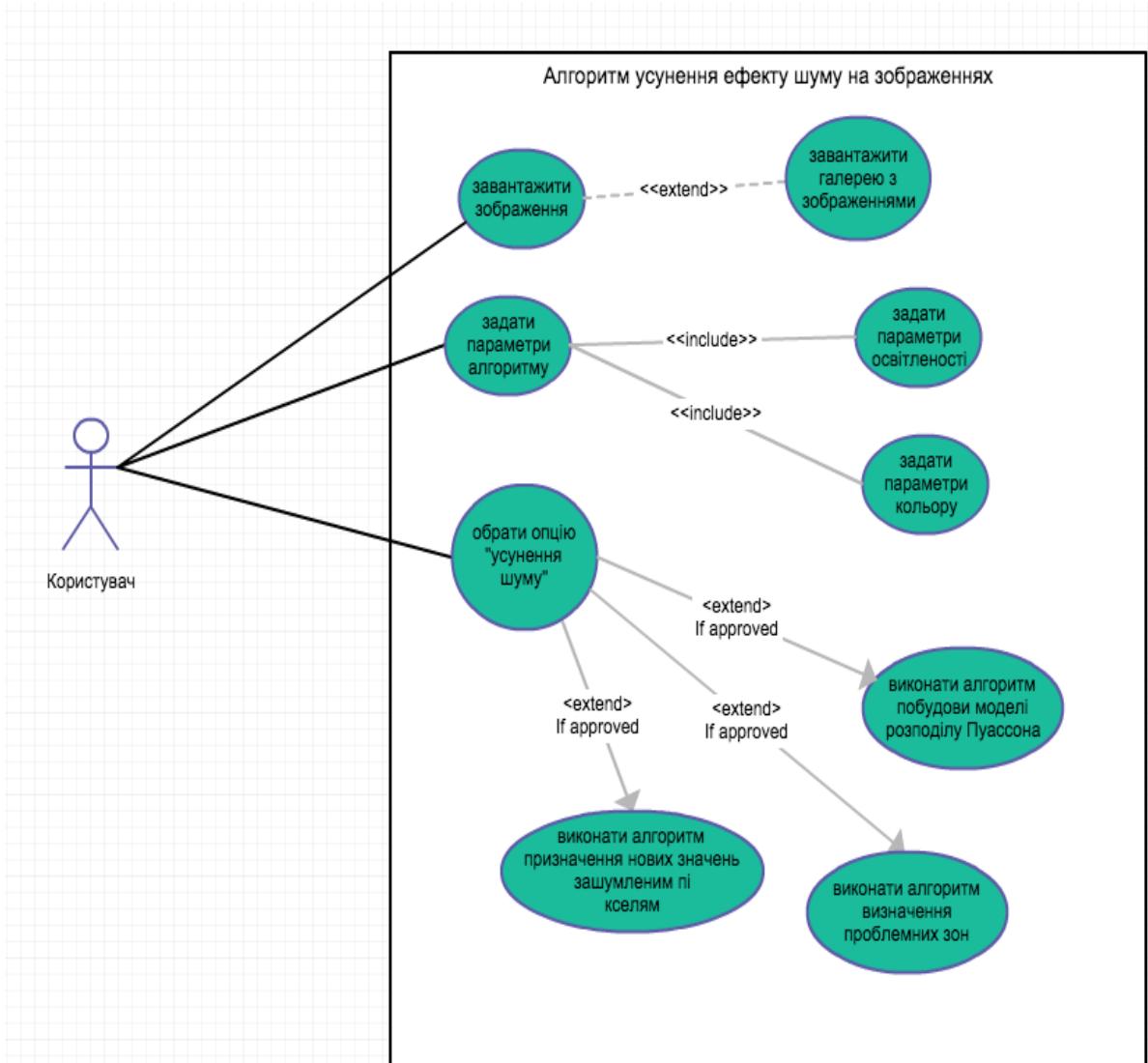


Рис. 3.1 Діаграма варіантів використання програми для усунення шуму

### **3.1.6. Обмеження проектування і реалізації**

1. Дослідження та програмну реалізацію алгоритму та інтерфейсу користувача слід завершити до грудня 2015 року.
2. Якість розв'язку отриманого з допомогою запропонованого методу, може бути гірша за якість розв'язку отриманого з допомогою інших алгоритмів.
3. Час знаходження розв'язку з допомогою запропонованого методу повинен бути меншим за час роботи інших алгоритмів.

### **3.1.7. Документація користувача**

Програмна реалізація алгоритму буде задокументована і описана з математичного боку у вигляді поясннюючої записки до магістерської дипломної роботи.

Пояснювальна записка міститиме:

1. Аналітичний опис існуючого стану досліджуваної проблеми.
2. Економічне обґрунтування доцільності реалізації програмної системи для усунення шуму на зображеннях.
3. Дослідження запропонованого алгоритму.
4. Інструкцію користувача.
5. Специфікацію вимог.
6. План забезпечення якості.

### **3.1.8. Характеристики системи**

**Усунення шуму на зображені**

**Опис:** Полягає у отриманні вхідного зображення, його обробці, та поверненні вихідного зображення.

**Пріоритет: Високий**

**Задання параметрів алгоритму**

**Опис:** Полягає у можливості налаштування параметра усунення шуму для

каналу освітленості та кольорових каналів.

Пріоритет: **Високий**

Вибір зображення

Опис: Полягає у забезпеченні діалового вікна для вибору зображення з галереї.

Пріоритет: **Середній**

Збереження зображення

Опис: Полягає у забезпеченні діалогу для експортування вихідного зображення у файл PNG/JPEG.

Пріоритет: **Середній**

### **3.1.9. Вимоги зовнішніх інтерфейсів**

#### **3.1.9.1. Користувальські інтерфейси**

У результаті дослідження програм-аналогів було виділено типові елементи інтерфейсу користувача, а саме: елемент регулювання сили усунення шуму в каналі освітленості, елемент регулювання сили усунення шуму в кольорових каналах, меню для завантаження та збереження зображення, поле для відображення зображення, вікно з інформацією про програму.

#### **3.1.9.2. Програмні інтерфейси**

Розробка програмної реалізації алгоритму буде здійснюватися на комп'ютері з ОС Mac OS X, розробка візуального інтерфейсу користувача буде здійснюватись для настільної системи Mac OS X та мобільної платформи Android.

### **3.1.10. Вимоги продуктивності**

Задача усунення шуму на зображення повинна виконуватись в режимі реального часу, тому до розробленого ПЗ ставляться високі вимоги швидкодії. Очікувана швидкість роботи алгоритму — 20 mpps для настільних платформ та 5 mpps для мобільних платформ.

## **3.2. План верифікації та валідації**

### **3.2.1. Мета**

Метою створення даного документу є визначення плану перевірки розроблюваного програмного продукту на відповідність до поставлених вимог.

### **3.2.2. Межі документу**

Даний документ описує план перевірки якості програмного продукту, що реалізує алгоритм усунення шуму на зображеннях, який реалізований для роботи на графічному процесорі та є модифікацією алгоритму Ridgelet-перетворення.

### **3.2.3. Стратегія тестування компонент та процедур**

- Процес тестування
  - протестовані мають бути всі розроблені методи
- Трасування вимог
  - позитивні тести підтверджують той факт, що всі функціональні вимоги до ПЗ виконані
- Компоненти, що будуть піддані тестуванню
  - shader\_fft
  - shader\_radon
  - shader\_wavelet
  - DenoiseAndroid
  - DenoiseCocoa
- План тестування
  - тестування коду здійснюється паралельно з розробкою функціональності
- Звіт про тестування
  - інформація про результати тестування заноситься у

## відповідний звіт про тестування

- Обмеження
  - проведення тестів передбачає попередні приготування тестових даних

### 3.2.4. План тестування

Назва тестового випадку	Опис кроків
Усунення шуму на зображенні	<ol style="list-style-type: none"><li>1. Зчитати вхідне зображення</li><li>2. Запустити програму на виконання</li><li>3. Вивести вихідне зображення</li></ol>
Задання параметрів алгоритму	<ol style="list-style-type: none"><li>1. Змінити значення візуального елемента сили шуму</li><li>2. Отримати зміни у вихідному зображення</li></ol>
Вибір зображення	<ol style="list-style-type: none"><li>1. Натиснути кнопку вибору зображення</li><li>2. Побачити вибране зображення у вікні перегляду</li></ol>
Збереження зображення	<ol style="list-style-type: none"><li>1. Натиснути кнопку збереження зображення</li><li>2. Отримати збережене зображення у форматі PNG/JPEG.</li></ol>

### 3.3. Звіт про тестування

Назва тестового випадку	Результат
Усунення шуму на зображенні	Успішно
Задання параметрів алгоритму	Успішно
Вибір зображення	Успішно
Збереження зображення	Успішно

### 3.4. Інструкція користувача

Дана програмна система призначена для усунення шуму на зображеннях.

Процес розв'язування задачі складається з таких кроків:

1. Завантаження вхідного зображення у форматі PNG/JPEG.
2. Задання параметрів алгоритму усунення шуму.
3. Наступним кроком є підбір параметрів алгоритму для отримання найкращого результату.
4. Збереження отриманого зображення в файл.

### **3.5. Обґрунтування технологій програмної реалізації**

Процесорна версія алгоритму для усунення шуму на зображеннях написана на C++, шейдерна на GLSL. Мобільна версія реалізована для ОС Android, десктопна на Cocoa.

Інтерфейс користувача для мобільних платформ реалізовано на андроїд.

Android [link] — це ОС і платформа для мобільних телефонів та планшетних комп'ютерів, створена компанією Google на основі ядра Linux. Підтримується альянсом Open Handset Alliance. Хоча Android базується на Linux, він дещо відокремлений від Linux-інфраструктури. Базовим елементом цієї операційної системи є реалізація Dalvik віртуальної машини Java, і все ПЗ базується на цій реалізації Java. Платформа легко пристосовується для використання VGA, бібліотек двовимірної і тривимірної графіки, розроблених на основі OpenGL ES 1.0-3.1 специфікації, традиційних інструментаріїв для смартфонів. Android підтримує відеокамери, фотоапарати, дотикові екрани, GPS, компаси, акселерометри, та прискорювачі 3D графіки.

Cocoa [link] — рідний об'єктно-орієнтований прикладний програмний інтерфейс для операційної системи Mac OS X виробництва компанії Apple. Cocoa складається в основному з двох бібліотек об'єктів Objective-C. Foundation Kit, часто просто Foundation, уперше з'явився в OpenStep. В Mac OS X він заснован на Core Foundation. Foundation являє собою об'єктно-орієнтовану бібліотеку загального призначення, яка забезпечує роботу з рядками та

значеннями, контейнери та ітерацію по них, розподілені обчислення, цикли обробки повідомлень та інші функції, не прив'язані прямо до графічного інтерфейсу. Application Kit або AppKit походить напряму від NeXTSTEP Application Kit. Він містить код, за допомогою якого програми можуть створювати графічний інтерфейс та взаємодіяти з ним. AppKit побудован на основі Foundation. Ключовий елемент архітектури Сосоа — це модель переглядів (views). Зовнішньо вона організована як звичайний фреймворк, але реалізована з використанням PDF для усіх операцій малювання, зхожого з PostScript. Крім того, це автоматично надає можливість виводу будь-якого перегляду на друк. Оскільки Сосоа обробляє обрізку, прокрутку, масштабування та інші типічні задачі відображення графіки, програміст звільняється від необхідності реалізовувати базову інфраструктуру і може зосередитися на унікальних аспектах розроблюваного застосунку.

### **3.6. План забезпечення якості програмного продукту**

#### **3.6.1. Мета**

Даний розділ призначений для загального огляду Плану забезпечення якості програмного продукту для розв'язання задачі комівояжера великої розмірності, що реалізує розроблений метод спільних ребер. План забезпечення якості програмного продукту покликаний визначити процедуру розробки програмного продукту, її розбиття на окремі етапи, а також забезпечити максимальну відповідність розробленого ПЗ до поставлених вимог.

Даний документ окреслює процеси, техніки та засоби, які будуть використані для забезпечення якості програмного продукту, впродовж процесу його розробки. Зокрема:

- Г Визначено коло відповідальних осіб за якість розроблюваного програмного продукту;
- Г визначено перелік процесів та робіт, які буде здійснено з метою забезпечення якості програмного продукту.

## **3.6.2. Управління**

### **3.6.2.1. Організація**

Даний програмний продукт розробляється як індивідуальний магістерський проект для отримання Диплому магістра з Програмної Інженерії. Відтак над реалізацією проекту працює лише один розробник, який відповідальний за імплементацію проекту. Окрім того контроль за роботою розробника здійснюється з боку наукового керівника, який на регулярній основі здійснює огляди виконаної роботи, контролює відповідність прогресу на роботами до складеного графіку, а також надає консультації у процесі розробки.

### **3.6.2.2. Завдання**

План забезпечення якості стосується всіх етапів розробки програмного продукту, аж до моменту виходу продукту у реліз. Можна виділити такі основні фази роботи над реалізацією програмної системи:

1. Фаза визначення вимог.

На даному етапі слід визначити які функції має виконувати система, тобто функціональні вимоги до програмної системи, вимоги продуктивності.

2. Фаза специфікації та дизайну

На даному етапі слід виділити основні вимоги до ПЗ які будуть виконані у окремий документ, а також визначити архітектуру майбутньої системи.

3. Фаза реалізації програмної системи

На даному етапі систему буде реалізовано на вибраній мові програмування

4. Фаза тестування розробленого програмного продукту

Впродовж кожного з цих етапів розробки програмного забезпечення слід:

- Г удосконалювати план забезпечення якості
- Г співпрацювати і отримувати відгуки

### **3.6.2.3. Ролі та обов'язки**

Науковий керівник: Кутельмах Роман Корнелійович

Розробник: Павлюк Ольга Василівна

Розробник відповідальна за наступні аспекти робот:

- Г Здійснення оцінки вартості реалізації проекту
- Г Визначення вимог до програмного продукту
- Г Розробка дизайну програмного продукту
- Г Розробка плану забезпечення якості
- Г Написання документації для програмного продукту
- Г Імплементація функціональності програмного продукту

Науковий керівник відповідальний за наступні аспекти роботи:

- Г Контроль за роботою розробника
- Г Консультації розробника

#### **3.6.2.4. Оцінка ресурсів, необхідних для забезпечення якості**

Забезпечення якості розроблюваного програмного продукту вимагає додаткових затрат. Зокрема з метою забезпечення дотримання описаних процесів слід докласти зусилля на тестування розроблюваного програмного продукту, на опанування програмних інструментів забезпечення якості, а також на проведення супутніх завдань з написання документації та її періодичного удосконалення. Економічна оцінка затрат на забезпечення якості програмного продукту буде описана у розділі з економіки.

### **3.6.3. Документація**

#### **3.6.3.1. Мета**

Буде розроблено наступні документи:

- Г Специфікацію вимог до робробленого ПЗ.
- Г План забезпечення якості програмного продукту.
- Г Опис архітектури програмної системи.
- Г Оцінка вартості імплементації програмної системи.
- Г Протокол тестування.

### **3.6.3.2. Мінімальні вимоги до документації**

Впродовж роботи над реалізацією програмної системи буде створено багато супутньої документації, тому на кожен такий документ накладається ряд вимог, зокрема, кожен документ має мати:

- Г назуву
- Г дату створення
- Г ім'я автора

Окрім того:

- Г вся документація має бути написана українською мовою;
- Г документація має відповідати шаблонам, наданим науковими керівниками;
- Г документація має бути оформленна відповідно до поставлених науковим керівником вимог.

Всі згадані вимоги однаково відносяться як до друкованих примірників, так і до їх електронних версій.

#### **3.6.3.2.1. Опис вимог до програмного забезпечення (SRD)**

Див. розділ 3.1.

#### **3.6.3.2.2. Опис дизайну програмного забезпечення (SDD)**

Розроблювана програмна система являє собою програмний комплекс, до якого входять:

- Г модуль FFT, реалізований для роботи на GPU
- Г модуль перетворення Радона, реалізований для роботи на GPU
- Г модуль вейвлет-перетворення, реалізований для роботи на GPU
- Г модуль інтерфейсу користувача для ОС Mac OS X
- Г модуль інтерфейсу користувача для ОС Android.

#### **3.6.3.2.3. Плани верифікації та валідації**

Див. розділ 3.2.

### **3.6.3.2.4. Звіт з верифікації та валідації**

Див. розділ 3.3.

### **3.6.3.2.5. Користувальська документація**

Див. розділ 3.4.

### **3.6.3.2.6. План конфігураційного управління (SCMP)**

Конфігурація програмної системи включає можливі конфігурації concorde при побудові початкових розв'язків та конфігурації реалізації LKH. Контроль за версіями документації здійснюється з допомогою розподіленої системи контролю версій git. Конфігурація середовища виконання здійснюється з допомогою інсталяційних скриптів, написаних на bash.

### **3.6.3.3. Інші документи**

Визначаються інші документи, що застосовуються в проекті розробки програмного забезпечення і програмних продуктів. Інші документи можуть включати наступне:

- ─ План процесу розробки
- ─ План інтеграції програмного забезпечення

### **3.6.4. Стандарти, практики, конвенцій, і метрики**

У процесі реалізації програмного продукту слід дотримуватися відповідних стандартів. Зокрема слід дотримуватися таких стандартів написання коду:

- ─ C++ Coding Standard
- ─ Android Guidelines.

Окрім того, слід дотримуватися таких стандартів оформлення документації:

- ─ IEEE 730 Standard for Software Quality Assurance Plans
- ─ IEEE Recommended Practice for Software Requirements Specifications.

### **3.6.5. Тестування**

Цей розділ визначає всі тести, не включені до етапу верифікації та валідації програмного забезпечення, охоплених SQAP і повинні вказувати на методи, які будуть використовуватися. Якщо окремий план тестування існує, то він повинна бути вказанний.

### **3.6.6. Звітність проблем і коригувальних дій**

Дляожної проблеми, що виникає в процесі роботи над системою має бути запропоновано її вирішення. Загалом можуть виникати такі види проблем:

**Проблеми документації:**

- Г неузгодженість різних документів;
- Г невідповідність шаблонам та стандартам, що вказані науковим керівником;
- Г помилки у документації;
- Г недостатній обсяг документації.

**Проблеми реалізації:**

- Г неповна реалізація функціональності;
- Г помилки у реалізації функціональності.

Вирішення проблем здійснюється зусиллями розробника. У випадку коли вирішення проблем зусиллями розробника неможливе, слід звернутися за консультацією до наукового керівника.

Звітування та документація виявлених проблем не здійснюється, тому що:

1. Над реалізацією системи працює лише один розробник.
2. Система не є великою.

У випадку, якщо проблема пов'язана з хибними архітектурними рішеннями чи неможливістю реалізувати описану функціональність, допускається зміна функціональних вимог до програмного продукту.

### **3.6.7. Засоби, методи і методики**

Для формулювання завдань дипломного проекту та контролю за прогресом їх виконання використовувався сервіс Redmine — вільний серверний веб-застосунок для управління проектами та відстежування помилок. До системи входить календар-планувальник та діаграми Ганта для візуального представлення ходу робіт за проектом та строків виконання. Redmine написано на Ruby on Rails, що означає легкість в розгортанні системи та її адаптації під конкретні вимоги. Для кожного проекту можна вести свої вікі та форуми.

### **3.6.8. Контроль середовища**

Середовище розробки програмного продукту легко переноситься завдяки типовим засобам розробки під MacOS/Android OS: IDE Xcode, Eclipse CDT, а також використанні системи контролю версій git.

### **3.6.9. Контроль розробника**

Розроблений програмний продукт буде перевірено на відповідність функціональним вимогам, описаним у розділі 3.2, а для розробленого алгоритму буде проведено тестування на тестовому наборі зображень з сайту розробників алгоритму Non-Local Means.

### **3.6.10. Збір коментарів, їх підтримка і збереження**

В процесі роботи над магістерською дипломною роботою буде розроблена документація, до якої увійдуть звіт про дипломне проектування, що включатиме специфікацію вимог до ПЗ, та план забезпечення якості програмного продукту.

У разі виникнення запитань чи програмних помилок з розробником можна буде зв'язатись за допомогою електронної пошти.

### **3.6.11. Навчання**

Розробник даного ПЗ пройшла такі курси:

- Г Алгоритми та структури даних
- Г Основи програмної інженерії
- Г Проектний менеджмент
- Г Моделювання та аналіз ПЗ
- Г Аналіз вимог до ПЗ
- Г Якість і тестування ПЗ
- Г Програмування для мобільних платформ
- Г Математичне забезпечення програмних систем

В результаті, розробник здобула необхідні знання для реалізації описаної системи та написання відповідної документації.

### **3.6.12. Управління ризиками**

Існує ймовірність виникнення ризиків під час реалізації алгоритму та програмного забезпечення. Їх необхідно пріоритетизувати та відслідковувати. Під час виконання дипломного проекту слід обговорювати проблеми, що виникають на певних етапах розробки.

## **3.7. Опис схеми реалізації алгоритму FFT на GPU**

Алгоритм FFT, реалізований на мові GLSL для виконання на графічному процесорі, є ітеративною версією рекурсивного алгоритму Кулі-Тьюокі для ДПФ для сигналу, довжина якого є степенем числа 2. Ітеративна версія цього алгоритму є “розгорткою” рекурсивної версії: спочатку виконуються кроки для останнього рівня рекурсії, а потім від передостаннього до першого.

Функція для зворотнього перетворення Фур’є відрізняється від функції прямого перетворення тільки параметром, що позначає знак множника, тому розроблено спільний інтерфейс для цих функцій.

У реалізованій для GPU версії алгоритму FFT виконується для частини зображення, її координати передаються як параметри алгоритму.

Ініціалізація початкової перестановки на GPU відбувається наступним чином :

1. Шейдер, що виконується на GPU, приймає текстуру з комплексним зображенням та текстуру з таблицею перестановки, згенерованою на CPU.

```
varying vec2 v_tex_coord;  
  
uniform sampler2D s_texture; // complex src  
  
uniform sampler2D s_texture2; // bit permutation table
```

Ще одним параметром шейдера є початкова координата частини зображення, для якої виконується перетворення Фур'є та розмір цієї частини:

```
uniform vec2 v_table_start;  
  
uniform float g_cell;
```

У GLSL відбувається зсув координат у центр пікселя, тому піксель з координатами (0, 0) переходить у (0.5/w, 0.5/h), де w та h — ширина та висота зображення відповідно. Тому передаються параметри sh0 та sh1 для обчислення абсолютних координат.

```
uniform float sh0; // big v_tex_coord shader shift: 1/2n  
uniform float sh1; // small v_tex_coord shader shift: 1/2*cell_size  
uniform float cell_size;
```

2. Функція main для шейдера, що виконує початкову перестановку у перетворенні Фур'є, виглядає наступним чином:

```
void main()  
{
```

Обчислюються абсолютні координати пікселя для таблиці перестановки:

```
vec2 v_table_shift = (v_tex_coord-sh0) - v_table_start;  
vec2 v_cell_start = g_cell*floor(v_table_shift/g_cell);  
vec2 v_cell = v_table_shift - v_cell_start;  
vec2 v_cell_local = v_cell/g_cell + sh1;
```

Обчислюються значення координат пікселя, який замінюється на даному кроці:

```
vec2 permute_idx_packed = (texture2D(s_texture2,
    vec2( v_cell_local.x, 0 )).xy)*255.0;
```

Текстура RGBA виділяє на кожен канал зображення тільки 8 біт, тому для координат в діапазоні 0..255 достатньо першого байта пікселя RGBA, а для координат, значення яких перевищує 255, необхідно зберігати цілу та дробову частину від ділення на 255 у першому та другому байті відповідно :

```
float permute_idx_local = ((permute_idx_packed.y*256.0 +
    permute_idx_packed.x)/cell_size)*g_cell;
```

У значення поточного пікселя записується значення пікселя з координатами, зчитаними з таблиці перестановки:

```
gl_FragColor = texture2D(s_texture, vec2(v_table_start +
    v_cell_start + vec2(permute_idx_local,v_cell.y) + sh0));
```

```
}
```

Після початкової перестановки відбувається виклик основної функції алгоритму, яка приймає номер кроку Кулі-Тьюкі та нормалізуючий множник в залежності від того, пряме це перетворення чи зворотнє:

```
int x = int(v_xy_int.x), y = int(v_xy_int.y);

int half_step = dst_step/2;

float is_negative = step(float(half_step), mod(float(x),
    float(dst_step)));

float sign_mult = 2.0*(0.5-is_negative);

int k = (x - (x/dst_step)*dst_step)-
    int(is_negative*float(half_step));

float exp_arg = M_PI*float(k)/float(half_step);

vec2 num_exp = vec2(cos(exp_arg), sin_sign*sin(exp_arg));
```

Нове значення комплексного числа встановлюється як на основі значень лівого та правого "сусідів", індекси яких залежать від кроку в циклі алгоритму.

```
int left_index = x - int(is_negative*float(half_step));  
int right_index = x + int((1.0 - is_negative)*float(half_step));  
vec2 res = t_left + mult_cpx(num_exp * sign_mult, t_right);  
res = res * norm_factor;  
gl_FragColor = pack_cpx_val(res);
```

### 3.8. Опис схеми реалізації алгоритму перетворення Радона на GPU

Для перетворення Радона, як і для перетворення Фур'є, реалізовано процесорну версію та версію для відеокарти. Оскільки, за теоремою Projection-Slice, рядки зображення у перетворенні Радона — це промені, що проходять через центр у перетворенні Фур'є, до яких застосовується зворотнє перетворення, то алгоритм для виконання на GPU для даного перетворення виглядатиме наступним чином:

1. Зчитування променів, що проходять через центр перетворення Фур'є.
2. Застосування до рядків отриманого зображення зворотнього перетворення Фур'є.

Функція main для шейдера, який зчитує промені у Фур'є-перетворенні зображення, виглядає наступним чином:

```
void main()  
{
```

Обчислюється кут для променя, який проходить через центр зображення:

```
float angle = (v_tex_coord.y - nsh.y) * M_PI;  
float ldist = (v_tex_coord.x - nsh.x);  
vec2 cc = vec2(0.5), sa = 0.5*vec2(cos(angle), sin(angle));
```

Знаходять значення точок перетину променя з межами зображення:

```
vec2 p1 = cc-sa, p2 = cc+sa;
```

Значення для даного пікселя встановлюється на основі лінійної інтерполяції координат відрізка на даному радіальному промені:

```
vec2 p = p1 + (p2-p1)*ldist + vec2(nsh.x);
```

```
gl_FragColor = texture2D(tex_fft, p );
```

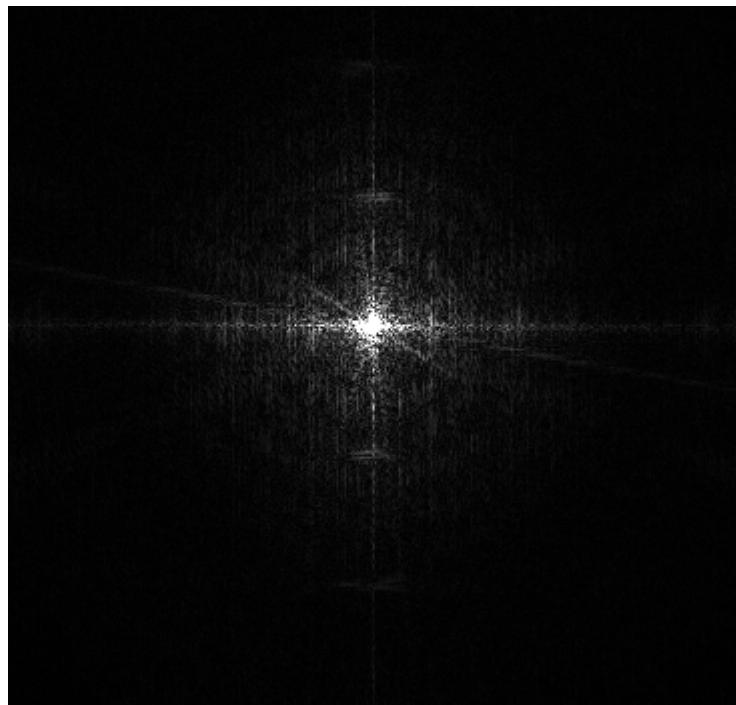
```
}
```

## **РОЗДІЛ 4. ДОСЛІДЖЕННЯ АЛГОРИТМУ SHADERRIDGELET ДЛЯ ЗАДАЧІ УСУНЕННЯ ШУМУ НА ЗОБРАЖЕННЯХ**

### **4.1. Дослідження результатів перетворення Радона**

Для тестування особливостей перетворення Фур'є було розроблено алгоритм для візуалізації цього перетворення в регіонах зображення розміру 256x256.

За допомогою алгоритму, описаного в п. 3.7, отримується зображення, на якому відображена магнітуда комплексних коефіцієнтів двовимірного перетворення Фур'є.



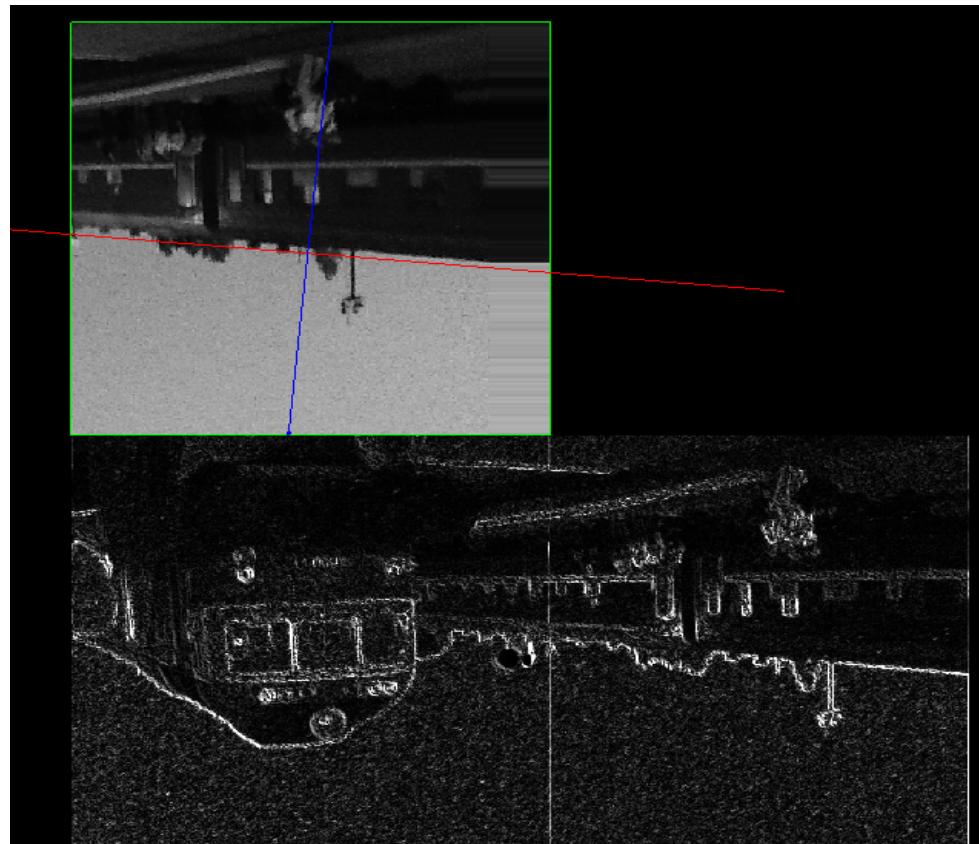
**Рис. 4.1.** Отримані лінії внаслідок застосування техніки виділення країв

### **4.2. Дослідження результатів перетворення Радона**

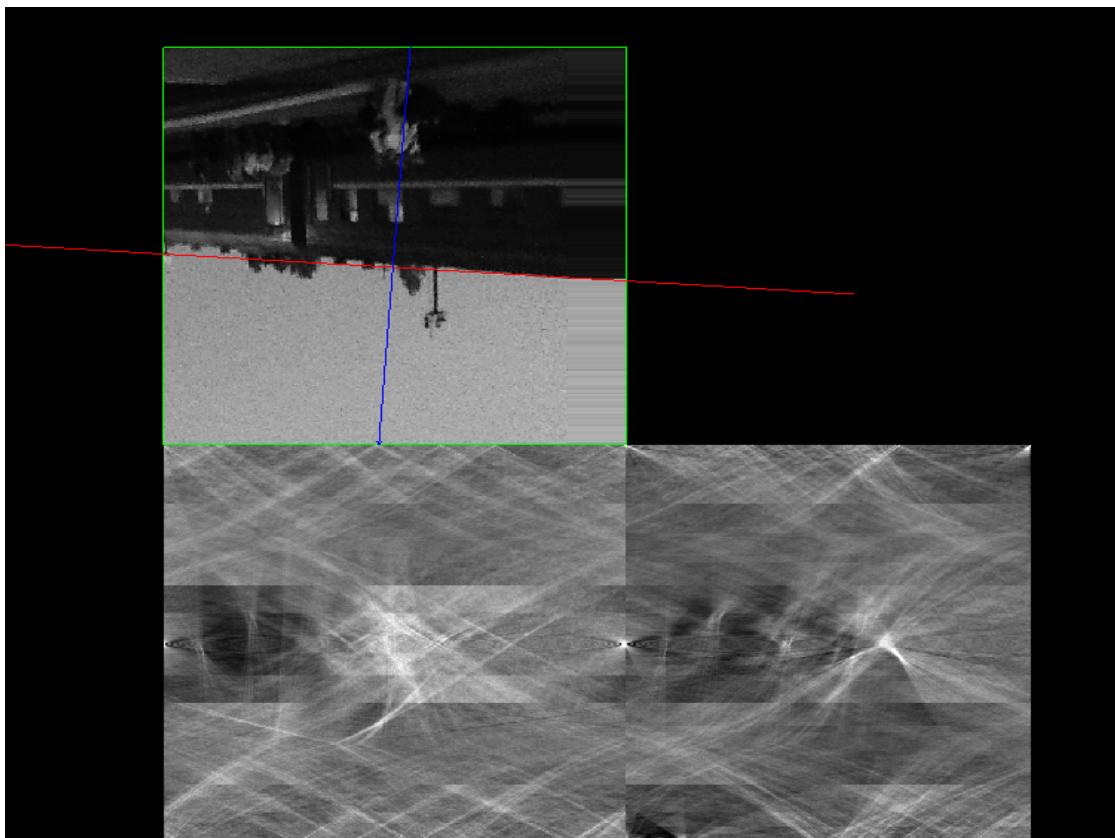
Для тестування коректності перетворення Радона було розроблено алгоритм для візуалізації цього перетворення в регіонах зображення розміру 256x256.

На зображеннях було застосовано виділення країв за допомогою

перетворення Фур'є для отримання максимальної інтенсивності вздовж ліній.



**Рис. 4.2.** Отримані лінії внаслідок застосування техніки виділення країв



**Рис. 4.3.** Перетворення Радона, у якому локальні максимуми відображають лінії

#### **4.3. Дослідження візуальної якості запропонованого методу**

Для дослідження візуальної якості розробленого алгоритму порівняємо його з існуючими аналогами для типових зображень розміру від 0.4 до 1 mp з середнім рівнем шуму 0.1.

Оскільки оригінальна реалізація методу Non-Local Means працює дуже довго (більше 10 хв) для зображень розміру більше 1mp, то на зображеннях розміру більше 1mp візуального порівняння запропонованого алгоритму ShaderRidgelet з алгоритмом NLM не проводилось.



**Рис. 4.4** Приклад оригінального зображення

З Рис. 4.4-4.9 отримано висновок, що алгоритм ShaderRidgelet дає гірші результати, ніж алгоритм Non-Local Means для кольорових зображень зі стандартним відхиленням шуму 0.1.



**Рис. 4.5** Приклад роботи алгоритму ShaderRidgelet



**Рис. 4.6** Приклад роботи алгоритму Non-Local Means



**Рис. 4.7** Приклад оригінального зображення



**Рис. 4.8** Приклад роботи алгоритму ShaderRigdelet



**Рис. 4.9** Приклад роботи алгоритму Non-Local Means

#### **4.4. Дослідження часу роботи запропонованого методу для усунення шуму ShaderRidgelet**

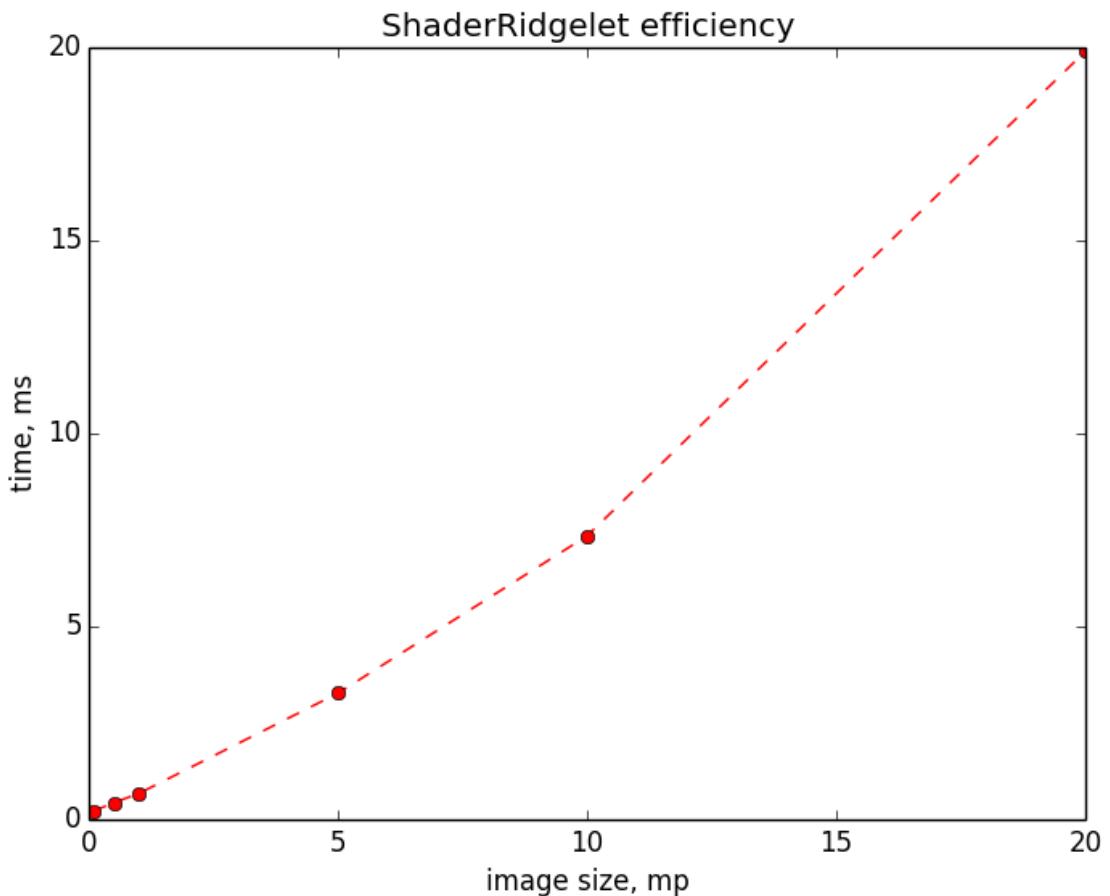
Як було розглянуто в п. 4.3, алгоритм Non-Local Means не підтримує режим реального часу для зображень середнього та великого розміру, тому для нього тестування швидкодії не проводились.

Таблиця 4.1

Порівняння часу роботи алгоритмів NLM та ShaderRidgelet

	0.1 mp	0.5 mp	1mp	5 mp	10mp	20 mp
NLM	23712ms	32307 ms	x	x	x	x
ShaderRidgelet	220 ms	435ms	671 ms	3271ms	7346ms	23902ms

На основі даних про час роботи алгоритмів, що порівнювались, отримано графік, що відображає різницю в обчислювальній складності, а також відмінності у швидкості виконання програмного коду на CPU та GPU.



З отриманого графіку обчислювальної складності можна зробити висновок про те, що експериметальні дослідження підтвердили належність обчислювальної складності запропонованого алгоритму до класу  $O(n * \log(n))$ .



## **РОЗДІЛ 5. ЕКОНОМІЧНА ХАРАКТЕРИСТИКА ПРОГРАМНОЇ СИСТЕМИ АЛГОРИТМУ УСУНЕННЯ ШУМУ НА ЗОБРАЖЕННЯХ**

### **5.1. Економічна характеристика програми для усунення шуму на зображеннях**

На даний час шум в цифрових зображеннях – це досить поширення проблема. Вона виникає внаслідок недостатнього освітлення та високої світлової чутливості (ISO) фотокамери. Найчастіше з цією проблемою зустрічаються користувачі мобільних платформ, для яких важливою є потреба забирання шуму з фотографій у режимі реального часу. Більшість існуючих програмних рішень на ринку мобільних платформ або працюють надто довго, забезпечуючи високу якість покращеного зображення, або дають незначне покращення за невеликий час.

Мета проекту – розробити алгоритмічне та програмне забезпечення для усунення шуму на зображеннях для мобільних платформ, що поєднує високу швидкість роботи та високу якість покращеного зображення.

Проведено дослідження програм-аналогів, з них найбільш конкурентно-спроможними є Topaz Denoise, NoiseWare та Dfine. Вони пропонують два типи програмних рішень – мобільні додатки та плагіни для популярних програм.

Програма, розроблена в рамках магістерського дослідження, поєднує швидкість та якість усунення шуму, тому зможе зайняти своє місце на ринку мобільних додатків.

### **5.2. Розрахунок витрат на розробку та впровадження програмної системи**

1) Витрати на розробку і впровадження програмного засобу ( $K$ ) визначаються як:

$$K_{\text{зас}} = K_1 + K_2 \quad (5.1)$$

де  $K_1$  – витрати на розробку програмного засобу, грн.,

$K_2$  – витрати на відлагодження і дослідну експлуатацію програмного засобу на ЕОМ, грн.

Витрати на розробку програмного засобу включають в себе:

1. Витрати на оплату праці розробників ( $B_{on}$ );
2. Єдиний соціальний внесок ( $B_{ев}$ );
3. Вартість додаткових виробів, що закуповуються ( $B_o$ );
4. Транспортно-заготівельні витрати ( $B_{mp}$ );
5. Витрати на придбання спецобладнання ( $B_o$ );
6. Накладні витрати ( $B_u$ );
7. Інші витрати ( $B_u$ ).

Для проведення розрахунків витрат на оплату праці необхідно визначити категорії працівників, які приймають участь в процесі проектування, їх чисельність, середньоденну заробітну плату спеціаліста відповідної категорії та трудомісткість робіт у людино-днях (людино-годинах).

У проекті беруть участь три особи: C++-програміст, OpenGL-програміст та інженер з тестування програмного забезпечення.

Їхня заробітна плата становить:

- Г C++-програміст – 10000 грн. (1 люд.);
- Г OpenGL-програміст – 15000 грн. (1 люд.);
- Г Інженер з тестування – 8400 грн. (1 люд.).

Трудомісткість робіт над проектом становить по два 24 людино-дні для програмістів та 10 людино-днів для інженера з тестування.

Середньоденна заробітна плата i-го розробника ( $\mathfrak{P}_{\text{д}_i}$ ) обчислюється за формулою:

$$\mathfrak{P}_{\text{д}_i} = \frac{\mathfrak{P}_i}{\Phi_M} \quad (5.2)$$

де  $\mathfrak{P}_i$  - основна місячна заробітна плата розробника i-ої спеціальності, грн.;

$\Phi_M$  – місячний фонд робочого часу, днів (24 дні).

$$\mathfrak{P}_{\text{д}_2} = \frac{10000}{24} = 416,6 \text{ грн.}$$

Середня заробітна плата C++-програміста становить 416,6 грн.

$$\mathfrak{P}_{\text{д}_1} = \frac{15000}{24} = 625 \text{ грн.}$$

Середня заробітна плата OpenGL-програміста становить 625 грн.

$$\mathfrak{P}_{\text{д}_1} = \frac{8400}{24} = 350 \text{ грн.}$$

Середня заробітна плата інженера з тестування становить 350 грн.

Розрахунок витрат на оплату праці усіх працівників обчислюємо за формулою:

$$B_{\text{оп}} = \sum_{i=1}^N n_i t_i \mathfrak{P}_{\text{д}_i} \quad (5.3)$$

де  $n_i$  – чисельність розробників проекту i-ої спеціальності;  $t_i$  – час, витрачений на розробку проекту працівником i-ої спеціальності,

дні;

$B_{\text{дн}}^i$  – денна заробітна плата розробника і-ої спеціальності, грн.;

$$B_{\text{он}} = 1 * 24 * 416,6 + 1 * 24 * 625 + 1 * 10 * 350 = 28500 \text{ грн.}$$

Розрахунок витрат на оплату праці розробників зведенено у табл. 5.1

Таблиця 5.1

Розрахунок витрат на заробітну плату

<i>№ з/ n</i>	<i>Спеціальність розробника</i>	<i>Кількість розробників</i>	<i>Час роботи , Дні</i>	<i>Денна заробітна плата , грн.</i>	<i>Витрати на оплату праці, грн.</i>
1	<i>C++-програміст</i>	1	24	416,6	10000
2	<i>OpenGL-програміст</i>	1	24	625	15000
3	<i>Інженер з тестування</i>	1	10	350	3500
<i>Разом</i>					28500

2) Витрати на оплату праці працівникам тягнуть за собою відрахування єдиного

соціального внеску  $B_{\text{ев}}$ , ставка якого залежатиме від класу професійного ризику розробників програмного забезпечення та визначатиметься у відсотковому співвідношенні від фонду оплати праці. Підприємство зобов'язане здійснити відрахування до Пенсійного фонду, згідно 2-го класу ризику – 36,77 %.

$$B_{\text{ев}} = 28500 * 0,3677 = 10479,45 \text{ грн.}$$

3) Витрати на додаткові вироби, що закупляються ( $B_o$ ) (папір, диски тощо) визначаються за їхніми фактичними цінами з врахуванням найменування, номенклатури та необхідної їх кількості в проекті. Вихідні дані та результати розрахунків занесені в табл. 5.2. Транспортно-заготівельні витрати ( $B_{mp}$ ) становлять 13% від суми витрат на додаткові вироби, що закупляються.

$B_{\Delta_1} = 68 \text{ грн}$  – вартість упаковки паперу формату А4.

$B_{\Delta_2} = 100 \text{ грн}$  – вартість канцелярських виробів.

Розраховуємо суму з урахуванням транспортно-заготівельних витрат:

$$B_o = 68 * 1 * 1,13 + 100 * 1 * 1,13 = 189,84 \text{ грн}$$

Таблиця 5.2

Розрахунок витрат на куповані вироби

<i>№ з/ п</i>	<i>Найменування купованих виробів</i>	<i>Марка, тип</i>	<i>Кількість на розробку, шт.</i>	<i>Ціна за одиницю , грн.</i>	<i>Сума витрат , грн.</i>	<i>Сума витрат з урахуванням транспортно- заготівельни- х витрат, грн.</i>
1	<i>Папір формату A4</i>	<i>Canon Yellow Label</i>	1	68	68	76,84
2	<i>Канцелярські вироби</i>	<i>Centropen</i>	1	100	100	113
<i>Разом</i>						<i>189,84</i>

Згідно проведених обчислень, усі здійснені витрати на додаткові вироби становлять 189,84 грн.

4) Витрати на придбання спецобладнання ( $B_o$ ) для проведення експериментальних робіт розраховуються в тому випадку, коли для розроблення та впровадження проектного рішення необхідне придбання

додаткових технічних засобів. Вартість спецобладнання для виконання конкретного проекту визначається на основі специфікації в їх потребі та фактичних цін з урахуванням транспортно–заготівельних витрат. Транспортно–заготівельні витрати ( $B_{tp}$ ) становлять 13% від суми витрат на придбання спецобладнання.

Вартість ноутбука Lenovo G580 1 шт. складає – 11000 грн. Розраховуємо суму витрат з урахуванням транспортно–заготівельних витрат, грн. та заносимо дані в табл. 5.3.

Розраховуємо суму витрат на ноутбук з врахуванням транспортно–заготівельних витрат:

$$B_w = 11000 * 1,13 = 12430 \text{ грн.}$$

Таблиця 5.3

*Розрахунок вартості спецобладнання*

<i>№ з/п</i>	<i>Найменування обладнання</i>	<i>Марка, тип</i>	<i>Кількість на проект, шт.</i>	<i>Ціна за одиницю, грн.</i>	<i>Сума витрат з урахуванням транспортно- заготівельних витрат, грн.</i>
<i>I</i>	<i>Ноутбук</i>	<i>Lenovo G580</i>	<i>1</i>	<i>11000</i>	<i>12430</i>
<i>Разом</i>					<i>12430</i>

Витрати на придбання спецобладнання становлять 12430 грн.

5) Накладні витрати ( $B_n$ ) проектних організацій передбачають витрати на управління, загальногосподарські, невиробничі витрати. Вони становлять 20-30% витрат на оплату праці.

Для розрахунку накладних витрат необхідно перемножити витрати на оплату праці на 25%:

$$B_n = 28500 * 0,25 = 7125 \text{ грн.}$$

6) Інші витрати ( $B_i$ ) – це усі ті витрати, які не були враховані в попередніх статтях витрат.

Їх розраховують за встановленими відсотками до витрат на оплату праці (становить 10%).

$$B_i = 28500 * 0,1 = 2850 \text{ грн.}$$

7) Щодо витрат на розроблення проектного рішення, то їх обчислюємо за формулою:

$$K_1 = 28500 + 10479,45 + 189,84 + 12430 + 7125 + 2850 = 61574,29 \text{ грн.}$$

$$K_1 = B_{on} + B_{eb} + B_d + B_{co} + B_n + B_{in}$$

Також потрібно обрахувати витрати на налагодження і дослідну експлуатацію системи. Їх можна визначити згідно із встановленою для цих обчислень формулою:

$$K_2 = S_{m.e.} \cdot t_{gid} \quad (5.5)$$

де  $S_{m.e.}$  – вартість однієї години роботи ПК, грн./год.;

$t_{gid}$  - кількість годин роботи ПК на налагодження програми, год.

При роботі ноутбук споживає 0,04 КВт/год., якщо вираховувати, що тариф на електроенергію становить 0,456 грн., то таким чином остаточна вартість однієї години роботи ПК ( $S_{m.e.}$ ) становитиме 0,01824 грн. (0,04 КВт/год.\* 0,456 грн.).

На написання та відлагодження системи було витрачено ( $t_{\text{від}}$ ) 58 днів ( $58*8 = 464$  год.), тому:

$$K_2 = 0,016416 * 464 = 7,61 \text{ грн.}$$

Результати усіх здійснених розрахунків зведені і записано в єдину табл. 5.4, де вказано кошторис витрат на розробку проектного рішення, а також окремі статті витрат.

$$K = 61574,29 + 7,61 = 61581,9 \text{ грн.}$$

Таблиця 5.4

Кошторис витрат на розробку проектного рішення

Найменування елементів витрат	Сума витрат, грн.
Витрати на розробку проектного рішення, у т.ч.:	61574,29
витрати на оплату праці	28500
сплата єдиного соціального внеску	10479,45
витрати на додаткові вироби, що закуповуються	189,84
витрати на придбання спецобладнання	12430
накладні витрати	7125
інші витрати	28500
Витрати на відлагодження і дослідну експлуатацію системи	7,61
Всього	61581,9

Сума витрат на розробку програмного продукту для усунення шуму на зображеннях становить 61581,9 грн. В цю суму входить заробітна плата розробників, відрахування у єдиний соціальний внесок, витрати на додаткові вироби, які купуються, витрати на спецобладнання, накладні витрати, а також інші витрати.

### **5.3. Визначення комплексного показника якості**

Комплексний показник якості ( $\Pi_{\text{я}}$ ) визначається шляхом порівняння показників якості проектованої системи із вибраного аналогу.

За аналог обирається продукт, що відповідає проектному рішенню (проектованій системі) по сфері застовування та функціональному призначенню і є широко представлений на обраному ринку.

В якості аналогу може бути вибраний варіант ручного виконання функцій, які автоматизуються в розроблених програмних засобах.

Для визначення  $\Pi_{\text{я}}$  використовується система показників технічного рівня і якості, яка містить в собі наступні групи, причому в кожній групі вказана в дужках мінімальна кількість показників:

1. показники призначення (3-4);
2. показники надійності (2-3);
3. показники безпеки (1-2);
4. патентно-правові показники (1-2);
5. ергономічні показники (1-2) тощо.

Комплексний показник якості проектованої системи визначаємо методом арифметичного середньозваженого з формули:

$$\Pi_{\text{я}} = \frac{\sum_{i=1}^m C_i q_i}{\sum_{i=1}^m C_i} \quad (5.6)$$

де  $m$  - кількість одиничних показників (параметрів), прийнятих для оцінки якості проектованої системи;

$q_i$  - коефіцієнт вагомості кожного з параметрів щодо їхнього впливу на технічний рівень та якість проектованої системи (встановлюється експертним шляхом), причому:

$$\sum_{i=1}^m q_i \leq 1,0$$

(5.7)

$C_i$  - часткові показники якості, визначені порівнянням числових значень одиничних показників проектованої системи і аналога за формулами:

$$C_i = \frac{\Pi_{npi}}{\Pi_{a_i}} \quad \text{або} \quad C_i = \frac{\Pi_{a_i}}{\Pi_{npi}}$$

(5.8)

де  $\Pi_{npi}$ ,  $\Pi_{a_i}$  - кількісні значення i-го одиничного показника якості відповідно проектованої системи і аналога.

В якості аналогу вибрано програму «NoiseWare» компанії ImaGenomic. Комплексний показник якості 0,86. Результати розрахунків наведені в табл. 5.5.

Таблиця 5.5

Визначення комплексного показника якості проектованої системи  
(аналогу)

Показники	Числове значення показників, бали		<i>Відносний показник якості, <math>C_i</math></i>	<i>Коефіцієнт вагомості <math>q_i</math></i>	$C_i \times q_i$
	<i>Аналог</i>	<i>Проект.</i> <i>Прогр.</i> <i>Продукт</i>			
Актуальність системи	6	9	1,50	0,3	0,45

Універсальність	6	8	1,33	0,2	0,27
Ступінь новизни	6	9	1,50	0,04	0,06
Доступ до системи	7	7	1	0,05	0,05
Продуктивність роботи системи.	7	7	1,00	0,15	0,15
Тривалість функціонування системи	6	9	1,50	0,09	0,135
Ймовірність помилки	1	2	2,00	0,02	0,04
Патентно-правовий	6	6	1	0,04	0,04
Зручність налаштування	6	6	1,00	0,05	0,05
Легкість експлуатації	5	5	1	0,06	0,06
<i>Всього</i>	56	68	12,83	1	1,30

Згідно цих показників, програма для усунення шуму на зображеннях має більше значення комплексного показника, ніж аналог.

#### 5.4. Визначення експлуатаційних витрат

При порівнянні програмних засобів в експлуатаційні витрати включають вартість підготовки даних ( $E_1$ ) і вартість годин роботи ПК ( $E_2$ ). Одноразові експлуатаційні витрати визначаються за формулою:

$$E_{\pi(A)} \lceil E_{1\pi(A)} \lceil E_{2\pi(A)} \quad (5.9)$$

де  $E_{\pi(A)}$  - одноразові експлуатаційні витрати на проектне рішення (аналог), грн.;

$E_{1\pi(A)}$  - вартість підготовки даних для експлуатації проектного рішення (аналогу), грн.;

$E_{2\pi(A)}$  - вартість машино-годин роботи ПК для проектного рішення (аналогу), грн.

Річні експлуатаційні витрати визначаються за формулою:

$$B_{[e]\pi[A]} \lceil E_{\pi[A]} * N_{\pi[A]} \quad (5.10)$$

де  $B_{[e]\pi[A]}$  – експлуатаційні річні витрати проектного рішення, грн.;

$N_{\pi[A]}$  - періодичність експлуатації проектного рішення (аналогу), разів/рік.

Вартість підготовки даних для експлуатації проектного рішення (аналогу) ( $E_1$ ) визначаються за формулою:

$$E_1 \lceil \int_{1}^N n_i \lceil t_i \lceil 3\pi_q \quad (5.11)$$

де  $i$  – номери категорій персоналу, які беруть участь у підготовці даних;

$n_i$  – чисельність співробітників  $i$ -ї категорії, осіб;

$t_i$  – трудомісткість роботи співробітників  $i$ -ї категорії, осіб;

$3\pi_q$  – середньогодинна ставка робітника  $i$ -ї категорії з врахуванням відрахувань єдиного соціального внеску, грн./год.

Середньогодинна ставка оператора визначається за формулою:

$$3\pi_{q_i} \lceil \frac{3\pi_{q_i}(1 \lceil b)}{\Phi_e} \quad (5.12)$$

де  $3\pi_{q_i}$  – основна місячна зарплата працівника  $i$ -ї категорії, грн.;

$b$  – коефіцієнт, який враховує єдиний соціальний внесок;

$\Phi_e$  – місячний фонд робочого часу, год.

$$3\pi_1 = \frac{10000 * (1 + 0,3677)}{24 * 8} = 71,23 \text{ грн.}$$

$$3\pi_2 = \frac{15000 * (1 + 0,3677)}{24 * 8} = 106,85 \text{ грн.}$$

$$3\pi_3 = \frac{8400 * (1 + 0,3677)}{24 * 8} = 59,83 \text{ грн.}$$

Розраховуємо вартість підготовки даних для експлуатації проектного рішення:

$$E_1 = 1 * 24 * 71,23 + 1 * 24 * 106,85 + 1 * 10 * 59,83 = 4872,22 \text{ грн.}$$

Вартість машино-годин роботи ПК для проектного рішення рівна 3 грн. Розраховуємо одноразові експлуатаційні витрати, вони становлять

$$E_{\pi} = 4872,22 + 3 = 4875,22 \text{ грн.}$$

Періодичність експлуатації проектного рішення становить 5 раз/рік. Враховуючи цей параметр розраховуємо річні експлуатаційні витати:

$$B_{(e)\pi} = 4875,22 * 5 = 24376,1 \text{ грн.}$$

Над проектом-аналогом працює 1 керівник проекту і по сумісництву програмний інженер, 1 програмний інженер та 1 тестер. Розраховуємо середньогодинну ставку для кожного працівника враховуючи що заробітна плата становить для керівника проекту 18000 грн., для програміста 10500 грн. та для тестера 7350 грн. Час роботи над проектом 24 дні.

$$3\pi_1 = \frac{18000 * (1 + 0,3677)}{24 * 8} = 128,22 \text{ грн.}$$

$$3\pi_2 = \frac{15000 * (1 + 0,3677)}{24 * 8} = 106,85 \text{ грн.}$$

$$3\pi_3 = \frac{7500 * (1 + 0,3677)}{24 * 8} = 53,43 \text{ грн.}$$

Розраховуємо вартість підготовки даних для експлуатації проектно-аналогового рішення:

$$E_{1,4} = 1 * 24 * 128,22 + 1 * 24 * 106,85 + 1 * 24 * 53,43 = 3077,33 + 2564,44 + 1282,22 = 6923,98 \text{ грн.}$$

Вхідні дані та отримані результати заносимо в таблицю 5.6.

Таблиця 5.6

Розрахунок витрат на підготовку даних для роботи на ПК

<i>Категорія персоналу</i>	<i>Чисельність співробітників i-ої категорії, чол.</i>	<i>Час роботи співробітників в i-ої категорії, год.</i>	<i>Середньогодинна ЗП співробітника i-ої категорії, грн.</i>	<i>Витрати на підготовку даних, грн.</i>
<i>Проектне рішення</i>				
<i>Програміст</i>	<i>1</i>	<i>176</i>	<i>106,85</i>	<i>2350,73</i>
<i>Тестер</i>	<i>1</i>	<i>80</i>	<i>53,43</i>	<i>534,26</i>
<i>Всього</i>				<i>2884,99</i>
<i>Аналог</i>				
<i>Керівник проекту</i>	<i>1</i>	<i>192</i>	<i>128,22</i>	<i>3077,33</i>
<i>Програміст</i>	<i>1</i>	<i>192</i>	<i>106,85</i>	<i>2564,44</i>
<i>Тестер</i>	<i>1</i>	<i>192</i>	<i>53,43</i>	<i>1282,22</i>
<i>Всього</i>				<i>6923,98</i>

Вартість машино-годин роботи ПК для проектно-аналогового рішення рівна 6,75 грн. Розраховуємо одноразові експлуатаційні витрати для проектно-аналогового рішення, вони становлять :

$$E_{(A)} = 6923,98 + 6,75 = 6930,73 \text{ грн.}$$

Періодичність експлуатації проектного рішення становить 5 разів/рік. Враховуючи цей параметр розраховуємо річні експлуатаційні витрати для проектно-аналогового рішення:

$$B_{(e)A} = 6930,73 * 5 = 34653,66 \text{ грн.}$$

### **5.5 Розрахунок ціни споживання проектного рішення**

Ціна споживання ( $\Pi_c$ ) – це витрати на придбання і експлуатацію проектного рішення за весь строк його служби:

$$\Pi_{C(\Pi)} \lceil \Pi_{\Pi} \lceil B_{(E)NPV} \quad (5.13)$$

де  $\Pi_{\Pi}$  – ціна придбання проектного рішення, грн.;

$B_{(E)NPV}$  – теперішня вартість витрат на експлуатацію проектного рішення (за весь час його експлуатації), грн.:

$$\Pi_{\Pi} \lceil K * \left[ 1 \lceil \frac{\Pi_p}{100} \right] \lceil (1 \lceil C_{ПДВ}) \lceil K_o \lceil K_k \quad (5.14)$$

де  $\Pi_p$  – норматив рентабельності (20%);

$K_o$  – витрати на прив'язку та освоєння проектного рішення на конкретному об'єкті, 1000 грн.;

$K_k$  – витрати на доукомплектування технічних засобів на об'єкті, 200 грн.;

$C_{ПДВ}$  – ставка податку на додану вартість (20 %).

$$\Pi_{\Pi} = 40260,17 * (1 + 0,2) * (1 + 0,2) + 1000 + 200 = 59174,64 \text{ грн.}$$

Теперішня вартість витрат на експлуатацію проектного рішення розраховується за формулою:

$$B_{(e)NPV} = \sum_{t=1}^T \frac{B_{(E)Nt}}{(1+R)^t} \quad (5.15)$$

де  $B_{(E)Nt}$  - річні експлуатаційні витрати в  $t$ -ому році, грн.;

$T$  - строк служби проектного рішення, років;

$R$  - річна ставка проценту банків, (16%).

Зважаючи на те, що  $B_{(e)Nt} = const$  протягом всього строку експлуатації розраховуємо  $B_{(E)NPV}$ :

$$\begin{aligned} \frac{B_{(E)Nt} * 1}{(1+R)^t} &= PV * \dot{B}_{(E)Nt} \\ B_{(E)NPV} &= \sum_{i=1}^T \dot{B}_{(E)Nt} \end{aligned} \quad (5.16)$$

Де  $PV$  - ставка дисконту на період  $T$ , яка визначається від річної процентної ставки  $R$  і періоду експлуатації  $T$ . При значенні  $R=16$ , ставка дисконту буде:

$T$	0	1	2	3	4
$PV$	1,0	0,87	0,76	0,66	0,57

Термін експлуатації становить 4 роки, тому  $PV=0,57$ . Теперішня вартість витрат на експлуатацію проектного рішення становить:

$$B_{(E)NPV} = PV * B_{(E)Nt} = 0,57 * 14439,96 = 8230,78 \text{ грн.}$$

Розраховуємо ціну споживання проектного рішення:

$$I_{C(N)} = 59\,174,64 + 8230,78 = 67\,405,42 \text{ грн.}$$

Ціна придбання аналогу – 65000 грн.

Визначаємо ціну споживання для аналогу. Визначаємо теперішню вартість витрат на експлуатацію аналогу. Термін експлуатації аналогу становить 4 роки, тоді  $PV=0,57$ . Оскільки  $B_{(e)N} = const$ , то значить:

$$B_{(E)NPV} = PV * B_{[E]A} = 0,57 * 34653,66 = 19752,58 \text{ грн.}$$

Тепер розраховуємо ціну споживання проекту аналогу:

$$\Pi_{C(A)} = 65\,000 + 19752,58 = 84752,58 \text{ грн.}$$

## 5.6. Визначення показників економічної ефективності

Якщо базою для порівняння обрані відповідні програмні засоби, в даному розділі розраховуються такі показники:

1) Показник конкурентоспроможності:

$$K_{KC} = \frac{\Pi_{\text{я}}}{\Pi_{\text{а}}} \quad (5.17)$$

$$K_{KC} = \frac{67\,405,42 * 1,3}{84752,58} = 1,03$$

2) Економічний ефект в сфері експлуатації (грн.):

$$E_{ekc} = B(e)a - B(e)n \quad (5.18)$$

$$E_{ekc} = 34653,66 - 14439,96 = 20213,70 \text{ грн.}$$

3) Економічний ефект в сфері проектування (грн.):

$$Enp = \Pi_a - \Pi_n \quad (5.19)$$

$$Enp = 65000 - 59\,174,64 = 5825,36 \text{ грн.}$$

Якщо  $Enp > 0$  та  $E_{ekc} > 0$ , то розраховується:

4) Додатковий економічний ефект в сфері експлуатації (грн.):

$$E_{\text{екзД}} \lceil \prod_{i=1}^T E_{\text{екз}} (1 \lceil R)^{T \lceil i}, \quad (5.20)$$

$$E_{\text{екзД}} = 20213,70 * (1,16^1 + 1,16^0) = 43661,58 \text{ грн.}$$

5) Додатковий економічний ефект в сфері проектування (грн.):

$$Enp_{\text{Д}} \lceil Enp \lceil (1 \lceil R)^T \quad (5.21)$$

$$Enp_{\text{Д}} = 5825,36 * (1 + 0,16)^2 = 7838,60 \text{ грн.}$$

6) Термін окупності витрат на програмний продукт(років)

$$Tok \lceil \frac{K}{E_{\text{екз}}} \quad (5.22)$$

$$Tok = \frac{40260,17}{20213,70} = 1,99$$

Результатуючі показники економічної ефективності зводяться в таблицю 5.7.

Таблиця 5.7

Показники економічної ефективності проектного рішення

Найменування показників	Одиниці вимірювання	Значення показників	
		Аналог	Проектне рішення
1	2	3	4
1. Капітальні вкладення	грн.		40260,17
2. Ціна придбання	грн.	65000	59 174,64
3. Річні експлуатаційні витрати	грн.	19752,58	8230,78
4. Ціна споживання	грн.	84752,58	67 405,42
5. Економічний ефект в сфері	грн.	-	20213,70

<i>Експлуатації</i>			
<i>6. Додатковий економічний ефект в сфері експлуатації</i>	грн.	-	43661,58
<i>7. Економічний ефект в сфері проектування</i>	грн.	-	5825,36
<i>8. Додатковий економічний ефект в сфері проектування</i>	грн.	-	7838,60
<i>9. Термін окупності витрат на проектування рішення</i>	Роки	-	1,99
<i>10. Коефіцієнт конкурентоспроможності</i>	-	-	1,03

Після проведення розрахунків бачимо, що розроблений програмний засіб має переваги в порівнянні з аналогом тому, що його вартість є меншою, ніж вартість аналога.

## 5.7. Висновки

В даному розділі було здійснено дослідження економічних показників доцільності розробки програмного продукту для усунення ефекту шуму на зображеннях для мобільних платформ.

Цільовою аудиторією виступають користувачі мобільних пристройів з високим параметром ISO фотокамери. При розробці програмного забезпечення для мобільних платформ основними параметрами є якість отриманого зображення та швидкість виконання алгоритму. Використання технологій C++ та OpenGL дозволяє досягти високої швидкості роботи алгоритму та знизити вартість супроводу системи. Розроблений алгоритм забезпечує високу якість отриманого зображення.

Ціна придбання продукту - 59 174,64 грн., що є нижчою за ціну аналогу. Проектне рішення дозволяє досягнути значний економічний ефект в сфері проектування та експлуатації. Коефіцієнт конкурентоспроможності – 1,03. Також програмний продукт для усунення шуму набрав більше значення комплексного показника якості, ніж аналог.

Аналіз отриманих даних показав, що проект повністю окупить себе менш ніж за 2 роки.

Проаналізувавши всю інформацію вище, можна зробити висновок, що програмний продукт усунення шуму для мобільних платформ буде затребуваним на ринку.

## ВИСНОВКИ

У рамках магістерського дипломного проекту досліджено проблему шуму в цифрових зображеннях. Розглянуто сучасні алгоритми вирішення даної проблеми: алгоритми базовані на патчах та базовані на вейвлетах. Проаналізовано недоліки та переваги кожного з них.

За результатами магістерської кваліфікаційної роботи подано до друку статтю «Investigation of Existing Image Denoising Algorithms», O. Pavliuk, R. Kutelmakh у Вісник Національного університету «Львівська політехніка»

Запропоновано алгоритм, який базується на існуючому алгоритмі Ridgelet-перетворення, проте має вдосконалену схему інтерполяції координат та розроблений для виконання на графічному процесорі, що збільшує його швидкодію у середньому в 10 разів відносно реалізації на процесорі.

Обчислювальна складність алгоритму:  $O(n * \log_2(n))$ , тобто така як у сучасних відомих алгоритмах. Якість алгоритму дещо нижча, ніж у існуючому де-факто стандарті Non-Local Means, але співрозмірна з алгоритмом Curvelet Transform, завдяки врахуванню локальних характеристик зображення та комбінації відомих підходів, яка враховує ці характеристики.

Розроблено алгоритм програмного забезпечення для усунення шуму на зображеннях та програму з інтерфейсом користувача.

## **СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ**

1. A. Buades, B. Coll, J.M Morel, “A review of image denoising algorithms with a new one”, Multiscale Modeling & Simulation, 2005
2. A. Buades, J.M Morel, “A non-local algorithm for image denoising”, Proc. of IEEE CVPR, 2005
3. Charles-Alban Deledalle, Joseph Salmon, Arnak Dalalyan, «Image denoising with patch based PCA: local versus global», Université Paris Diderot-Paris, Paris, France, 2008
4. Thomas Brox, Oliver Kleinschmidt, Daniel Cremers “Efficient Nonlocal Means for Denoising of Textural Patterns”, Saarland University, Saarbrücken, Germany, 2007
5. Christopher E. Heil, David F. Walnut, “Continuous and Discrete Wavelet Transforms”, SIAM Rev., 31(4), 628–666, 1989
6. E. J. Candès and D. L. Donoho, “Curvelets – a surprisingly effective nonadaptive representation for objects with edges”, Saint-Malo, 1999
7. D. L. Donoho, “Digital ridgelet transform via rectopolar coordinate transform”, Technical report, Stanford University, 1998.
8. D.L. Donoho and M.R. Duncan. Digital curvelet transform: strategy, implementation and experiments. In H.H. Szu, M. Vetterli, W. Campbell, and J.R. Buss, editors, Proc. Aerosense 2000, Wavelet Applications VII, volume 4056, pages 12–29. SPIE, 2000
9. J.-L. Starck, E. Candès, and D.L. Donoho. The curvelet transform for image denoising. IEEE Transactions on Image Processing, 11(6):131–141, 2002
10. J.-L. Starck, E. Candes, and D.L. Donoho. Astronomical image representation by the curvelet transform. Astronomy and Astrophysics, 398:785–800, 2003

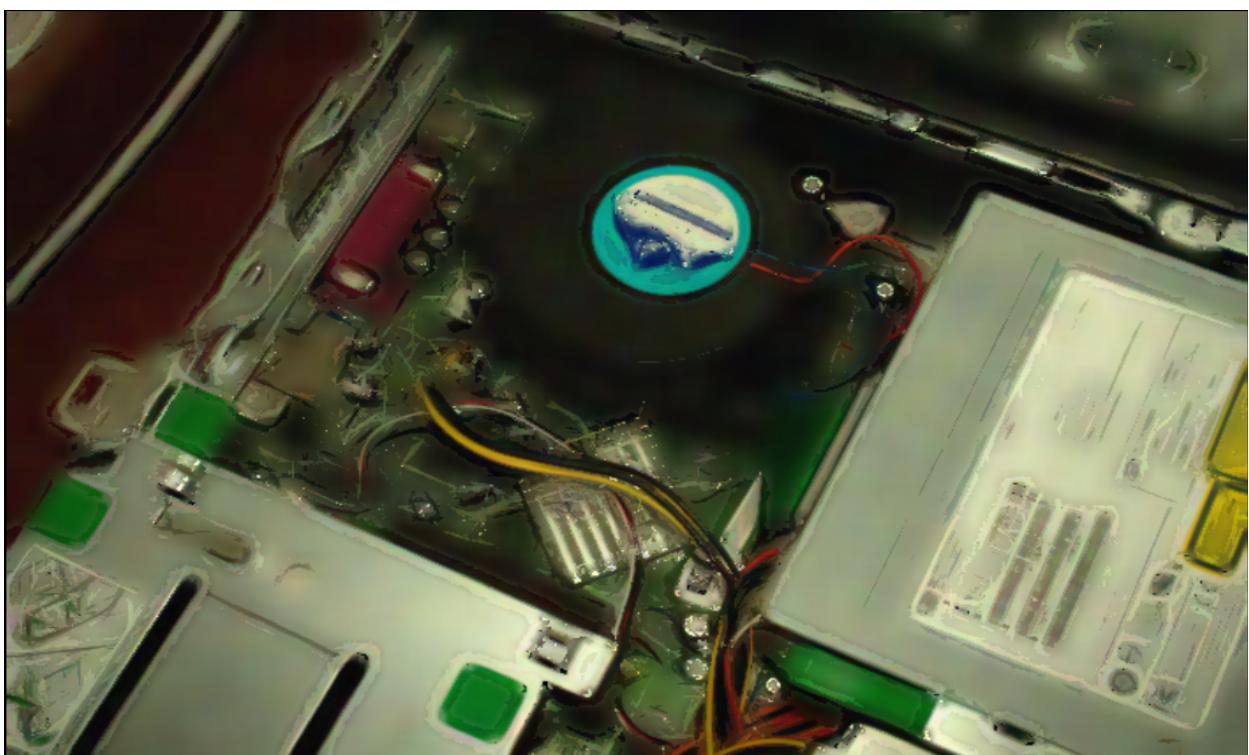
11. The Industry's Foundation for High Performance Graphics [Електронний ресурс] – Режим доступу: <https://www.opengl.org/>
12. Non-Local Means Denoising Image Archive [Електронний ресурс] – Режим доступу: [http://demo.ipol.im/demo/bcm\\_non\\_local\\_means\\_denoising/archive](http://demo.ipol.im/demo/bcm_non_local_means_denoising/archive)
13. GLSL Specification [Електронний ресурс] – Режим доступу: <https://www.opengl.org/registry/doc/GLSLangSpec.4.40.pdf>
14. Projection-Slice Theorem [Електронний ресурс] – Режим доступу: [https://en.wikipedia.org/wiki/Projection-slice\\_theorem](https://en.wikipedia.org/wiki/Projection-slice_theorem)
15. Topaz Denoise [Електронний ресурс] – Режим доступу: <https://www.topazlabs.com/denoise>
16. Download Topaz Denoise [Електронний ресурс] – Режим доступу: <https://www.topazlabs.com/downloads>
17. NoiseWare [Електронний ресурс] – Режим доступу: <https://www.imagenomic.com/nw.aspx>
18. Neat Image [Електронний ресурс] – Режим доступу: <http://www.neatimage.com/mac/standalone/download.html>
19. Photo Ninja [Електронний ресурс] – Режим доступу: <http://www.picturecode.com/download.php>
20. Dfine (Nick Collection) [Електронний ресурс] – Режим доступу: <https://www.google.com/nikcollection/products/dfne/>
21. Perfectly Clear [Електронний ресурс] – Режим доступу: <http://www.athentech.com/>
22. Review: Eliminate Photo Noise with Noiseware for iPhone [Електронний ресурс] – Режим доступу: <http://lifeinlof.com/2015/03/12/review-why-you-need-noiseware-for-iphone/>

**ДОДАТОК А. Результати роботи алгоритму для зображень зі значенням  
шуму  $\sigma=0.18$**

Оригінальне зображення



Приклад роботи алгоритму



**ДОДАТОК Б. Результати роботи алгоритму для зображень зі значенням  
шуму  $\sigma=0.18$**

Оригінальне зображення



Приклад роботи алгоритму



## ДОДАТОК В. Програмний код алгоритму DWT

```
// Wavelet.cpp

#include "Wavelet.h"

D4wavelet::D4wavelet()
{
    double denum = 4*sqrtf(2);

    h0 = (1+sqrtf(3))/denum;
    h1 = (3+sqrtf(3))/denum;
    h2 = (3-sqrtf(3))/denum;
    h3 = (1-sqrtf(3))/denum;

    g0 = h3;
    g1 = -h2;
    g2 = h1;
    g3 = -h0;
}

void D4wavelet::waveletTransformForward( std::vector<float>& x, int s0, int n )
{
    std::vector<float> tmp = x;
    int i = 0;

    for (int j = 0; j < n-3; j += 2)
    {
        tmp[s0+i] = x[s0+j]*h0 + x[s0+j+1]*h1 + x[s0+j+2]*h2 + x[s0+j+3]*h3;
        tmp[s0+i+n/2] = x[s0+j]*g0 + x[s0+j+1]*g1 + x[s0+j+2]*g2 + x[s0+j+3]*g3;
        i++;
    }

    tmp[s0+i] = x[s0+n-2]*h0 + x[s0+n-1]*h1 + x[s0+0]*h2 + x[s0+1]*h3;
    tmp[s0+i+n/2] = x[s0+n-2]*g0 + x[s0+n-1]*g1 + x[s0+0]*g2 + x[s0+1]*g3;
    x = tmp;
}

void D4wavelet::waveletTransformBackward( std::vector<float>& x, int s0, int n )
{
    std::vector<float> tmp = x;
    //      last smooth val  last coef.  first smooth  first coef
    tmp[s0+0] = x[s0+n-1]*h2 + x[s0+n*2-1]*g2 + x[s0+0]*h0 + x[s0+n]*g0;
```

```

tmp[s0+1] = x[s0+n-1]*h3 + x[s0+n*2-1]*g3 + x[s0+0]*h1 + x[s0+n]*g1;

int j = 2;

for (int i = 0; i < n-1; i++)
{
    //      smooth val      coef. val      smooth val      coef. val
    tmp[s0+j++] = x[s0+i]*h2 + x[s0+i+n]*g2 + x[s0+i+1]*h0 + x[s0+i+n+1]*g0;
    tmp[s0+j++] = x[s0+i]*h3 + x[s0+i+n]*g3 + x[s0+i+1]*h1 + x[s0+i+n+1]*g1;
}

x = tmp;
}

std::vector<float> WaveletPyramid::denoise(float thresh)
{
    if(m_levels == 0) return m_x;

    int N = int(m_x.size());
    int n_cur = N;

    for(int level = 0; level < m_levels; ++level)
    {
        assert(n_cur == pow2ceil(n_cur));
        for(int l0 = 0; l0 < (N/n_cur); ++l0)
        {
            waveletTransformForward(m_x, n_cur*l0, n_cur);
            // LOGI("wavelets FORWARD x[%d : %d]", n_cur*l0, n_cur);
        }
        n_cur /= 2;
    }

    n_cur *= 2;

    for(int level = 0; level < m_levels; ++level)
    {
        assert(n_cur == pow2ceil(n_cur));
        // thresh *= logf(n_cur);
        for(int l0 = 0; l0 < (N/n_cur); ++l0)
        {
            int start = n_cur*l0, half_len = n_cur/2;
            // LOGI("cut high-freq coefficients for x[%d : %d]", start+half_len, start+n_cur);
        }
    }
}

```

```

        bool cut_high_not_low_frequencies = false;
        if(cut_high_not_low_frequencies)
        {
            for(int high_p = start+half_len; high_p < start+n_cur; +
+high_p)
                if(fabs(m_x[high_p]) < thresh) m_x[high_p] = 0;
        }
        else
        {
            for(int low_p = start; low_p < start+half_len; ++low_p)
                if(fabs(m_x[low_p]) < thresh) m_x[low_p] = 0;
        }
        waveletTransformBackward(m_x, start, half_len);
        // LOGI( "wavelets BACKWARD for x[%d : %d]", n_cur*10, n_cur);
    }
    n_cur *= 2;
}
return m_x;
}

```