Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

РЕАЛІЗАЦІЯ АЛГОРИТМІВ ЦИФРОВОЇ ОБРОБКИ ЗОБРАЖЕНЬ МОВОЮ SWIFT

Текстова частина до кваліфікаційної роботи за спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи
ас. Корнійчук М. А.
(прізвище та ініціали)

_____
(підпис)
“____” _____ 2023 р.

Виконала студентка
Семенко Е. О.
(прізвище та ініціали)
“____” _____ 2023 р.

Київ 2023

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.т.н

Жежерун О. П.

(прізвище та ініціали)

_____

(підпис)

“____” _____ 2023 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на кваліфікаційну роботу

студентці Семенко Еліні Олександрівні 4-го курсу факультету інформатики

ТЕМА Реалізація алгоритмів цифрової обробки зображень мовою Swift

Зміст ТЧ до курсової роботи:

Анотація

Вступ

1. Дослідження та аналіз предметної області. Цифрова обробка зображень: огляд, процеси та застосування.

2. Інструменти цифрової обробки зображень в програмах, написаних мовою Swift.

3. Структура бібліотеки та реалізовані алгоритми.

Висновки

Список використаної літератури

Додатки (за необхідністю)

Дата видачі “____” _____ 2023 р.

Керівник ас. Корнійчук М. А.

_____

(підпис)

Завдання отримала

_____

(підпис)

Календарний план виконання курсової роботи

Тема: Реалізація алгоритмів цифрової обробки зображень мовою Swift

№ п/п	Назва етапу проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу	01.11.2022	
2.	Аналіз технічної літератури за темою	15.01.2023	
3.	Реалізація алгоритмів цифрової обробки зображень	01.03.2023	
4.	Написання текстової частини до кваліфікаційної роботи	15.04.2023	
5.	Коригування та оформлення текстової частини роботи	01.05.2023	
6.	Створення слайдів для доповіді та написання доповіді	15.05.2023	
7.	Остаточне оформлення роботи та слайдів	24.05.2023	
8.	Захист кваліфікаційної роботи	31.05.2023	

Студентка Семенко Е. О. _____

Керівник Корнійчук М. А. _____

“____” _____ 2023 р.

Зміст

Анотація

Робота присвячена аналізу галузі цифрової обробки зображень та розробці бібліотеки, що містить імплементацію алгоритмів цифрової обробки зображень мовою Swift. Результатом роботи є фреймворк з обробки зображень, що розширює та доповнює існуючі інструменти. Він містить допоміжні функції для розробників, набори згорткових фільтрів та операторів, та програми-генератори шуму. Фреймворк може бути інтегрований в додатки, написані мовою програмування Swift, та розширений за потреби.

Ключові слова: цифрова обробка зображень, згорткові фільтри, алгоритми з обробки зображень.

Вступ

Процеси цифрової обробки зображень є предметом комплексних досліджень та розробок протягом останніх десятиліть. Ця широка галузь охоплює такі теми, як власне обробка зображень, аналіз, класифікація та сприйняття зображень або відео. Дані процеси знайшли широке застосування в медичному моделюванні та діагностиці, автоматизації виробничих процесів та машинному навчанні. Крім цього, технології обробки зображень займають центральне місце в багатьох інших сферах, включаючи цифрову фотографію, відеоконференції та цифровізацію документів [1].

На даний момент, завдяки різноманіттю інструментів та технологій, цифрова обробка зображень може виконуватися на різних пристроях, в залежності від потреб сфери застосування. Такими пристроями можуть бути мобільні телефони, планшети та персональні комп'ютери. Процеси обробки зображень можуть виконуватися як на графічному процесорі пристроїв (GPU) [2], так і на центральному (CPU) [3].

Метою даної роботи є дослідження галузі цифрової обробки зображень та процесів, що вона включає; огляд наявних інструментів цифрової обробки зображень на пристроях з операційними системами iOS та MacOS, їхню структуру та функції; розробка фреймворку з цифрової обробки зображень, що містить зручні розширення функцій наявних інструментів та імплементацію алгоритмів з обробки цифрових зображень та генерації текстур, що відсутні в наявних бібліотеках, мовою програмування Swift.

Об'єктом дослідження є галузь цифрової обробки зображень, її процеси та складові.

Предметом дослідження є інструменти обробки зображень в програмах написаних мовою Swift.

Результатом роботи є розроблений фреймворк з функціями цифрової обробки зображень з можливістю інтеграції в програми для пристроїв з операційними системами iOS та MacOS. Фреймворк містить розширення

існуючих іструментів, а також додаткові алгоритми з обробки зображень, що наразі відсутні в існуючих бібліотеках.

Робота складається з 3-х розділів:

Перший розділ присвячений дослідженню предметної області. Розглядається галузь цифрової обробки зображень та процеси, що вона включає.

Другий розділ присвячений огляду наявних інстументів цифрової обробки зображень на пристроях з операційними системами iOS та MacOS. Розглядаються доступні бібліотеки та мови програмування, способи їхнього застосування та інтеграції процесів обробки зображень в програмні продукти, розроблені мовою Swift.

Третій розділ роботи містить огляд методів та інструментів використаних під час розробки та розповсюдження фреймворку з обробки зображень.

Розділ 1. Цифрова обробка зображень

1.1 Огляд галузі цифрової обробки зображень

Цифрова обробка зображень, також відома як комп'ютерна обробка зображень - це процес перетворення сигналів зображення на цифрові сигнали та подальша їх обробка за допомогою комп'ютера [4]. Така обробка зазвичай включає в себе візуальну зміну зображення або виділення деяких властивостей/характеристик зображення з метою їхнього наступного аналізу. Наразі межа між технологіями обробки зображень, аналізу зображень та комп'ютерного зору не є чітко визначеною, отже ці поняття варто розглядати в сукупності. Згідно з Гонсалесом [5], їх також можна умовно поділити на низькорівневі, середньорівнені та високорівневі процеси.

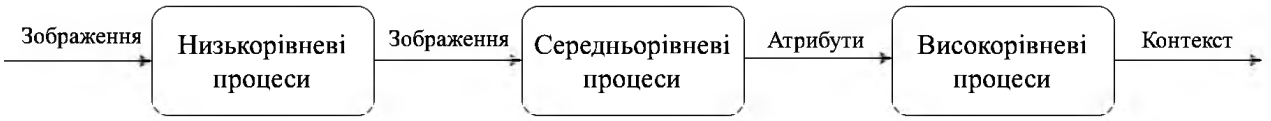

Рисунок 1.1 — Поділ процесів обробки зображень на рівні

Процеси низького рівня можуть включати примітивні операції, такі як зменшення шуму або підвищення контрастності зображення, тобто його первинну обробку. Низькорівневий процес характеризується також тим, що його вхідними та вихідними даними є зображення.

Процеси середнього рівня можуть включати такі завдання, як сегментація (поділ зображення на області), опис цих областей, і класифікація (розпізнавання) окремих об'єктів. Вхідними параметрами середньорівневих процесів є також зображення, однак вихідними є атрибути цих зображень (наприклад, кількість унікальних кольорів або контури зображення).

Процеси високого рівня включають роботу з більш складними характеристиками зображень (наприклад, визначення тематики зображення або класифікація взаємодії його елементів). Такі характеристики є більш загальними

та не мають чітко визначеної структури, отже застосовні до різноманітних зображень або відео-кадрів. Обробка зображень високого рівня може бути реалізована за допомогою алгоритмів штучного інтелекту, таких як нейронні мережі.

Для того, щоб повною мірою охопити усі процеси, що лежать в основі цифрової обробки зображень, варто визначити поняття зображення.

Будь-яке зображення можна визначити як двовимірну функцію $f(x, y)$, де x і y — просторові координати, а амплітуда функції f у будь-якій парі координат (x, y) показує інтенсивність кольору (intensity) або рівень сірого (grey level) в конкретній точці зображення [5]. У випадку кольорового зображення, значення функції в певній точці визначається як поєднання значень інтенсивності трьох кольорів: червоного, зеленого та синього, за формулою *(1.1)*:

$$(f_r(x, y), f_g(x, y), f_b(x, y)) \qquad (1.1)$$

Якщо для деякого зображення координати (x, y) та значення функції в будь-якій точці є скінченними дискретними величинами, таке зображення називається цифровим зображенням [5]; якщо ці значення є безперервними, воно називається аналоговим зображенням [6]. Отже, цифрове зображення можна розглядати як двовимірну матрицю деяких елементів, кожен з яких містить дані про колір та рівень прозорості в певній точці. Такі елементи називаються пікселями. В стандартному представленні зображень пікселі є рівномірно розташованими у двовимірній матриці, однак допустимі також і інші представлення. Наприклад, в рідкокристалічних екранах (LCD), три основних кольори знаходяться в різних місцях піксельної сітки в шаховому порядку [7]. Форма пікселя також може різнитися: у комп'ютерних моніторах пікселі мають квадратну форму, в інших системах, таких як широкоекранний формат цифрового відеостандарту 601, форма пікселя є прямокутною [8]. Залежно від системи кольорів, для визначення кожного кольорового компонента пікселя може бути виділена різна кількість пам'яті. Для бінарного зображення (зі значеннями кольорів 0 та 1), виділена на піксель память складає 1 біт; для

монохромного зображення, пікселі якого зображають відтінки сірого, виділена пам'ять складає 8 біт, що означає, що максимальна кількість кольорів, які можна відобразити одночасно, становить 256 або 2^8. Для кольорових зображень в більшості систем виділена пам'ять складає 24 біт (по байту на кожен кольоровий компонент) [9].

1.2 Процеси в основі цифрової обробки зображень

Оскільки процес повної цифрової обробки зображень є досить широким та різноманітним, має місце його поділ на основні фази або розділи наступним чином [5].

Отримання зображення (Image Aquisition). На цьому етапі отримуються та оцифровуються сигнали зображення, створені датчиком всередині камери. Також може відбуватися початкова обробка зображення, наприклад, зміна його розміру, для зручності подальшої роботи.

Попередня обробка (Image Pre-processing). Цей етап включає покращення та відновлення зображення. Отримані під час попереднього процесу зображення часто мають дефекти або спотворення, що спричинені обмеженням пропускної здатності або помилками в процесі формування зображення. Відповідно, первинна обробка є дуже важливою для коректного результату роботи системи. Процес попередньої обробки полягає у зменшенні шуму, коригуванні контурів зображення та усунення розмитостей.

Сегментація (Image Segmentation). На даному етапі виконується поділ вхідного зображення на складові частини об'єкта. Процес поділу зображення на кілька непересічних областей відбувається таким чином, що кожна область є однорідною, а об'єднання двох суміжних областей не є однорідним.

Вибір ключових об'єктів (Feature Selection). На цьому кроці виконується вибір ключових об'єктів зображення, придатних для подальшої комп'ютерної обробки. Методи вибору таких об'єктів використовуються для зменшення кількості вхідних змінних моделі класифікації шляхом усунення зайвих або нерелевантних областей.

Розпізнавання та інтерпретація (Recognition and Interpretation). Процес розпізнавання об’єкту складається з призначення йому певної мітки (тегу) відповідно до інформації, наданої його дескриптором. Інтерпретація полягає в наданні значення або контексту розпізнаному об’єкту.

З урахуванням процесів, з яких складаються описані етапи обробки зображень, їх також можна поділити за рівнем складності. Отримання то попередня обробка зображення є низькорівневими процесами; сегментація та вибір ключових об’єктів є процесами середнього рівня; розпізнавання та інтерпретація є відповідно процесами високого рівня.

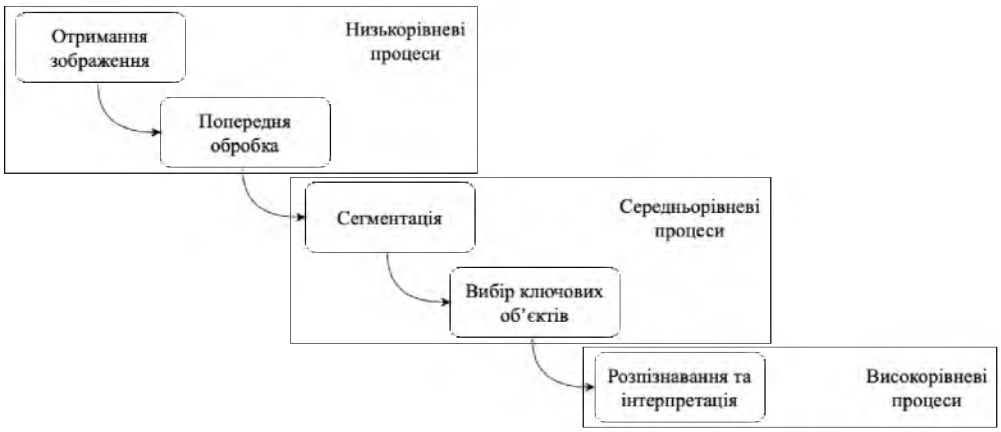

Рисунок 1.2 — Основні етапи обробки зображень

1.3 Застосування технологій цифрової обробки зображень

Технологія цифрової обробки зображень вперше була застосована в медичній сфері та продовжує бути корисною в таких медичних дослідженнях, як ультразвукова діагностика, рентгенівська ангіографія або комп’ютерна томографія. Цифрова обробка зображень відіграє дуже важливу роль у діагностиці захворювань та їхньому лікуванні, наприклад, визначенні складності переломів або первинній діагностиці раку. Процес полягає у використанні технології накладання зображення для проведення неінвазивних тестів [1].

Біологічна, а зокрема ботанічна сфера також використовує технологію цифрової обробки зображень для ідентифікації та аналізу шкідників і хвороб, а

також вилучення їхніх характеристик та особливостей [1]. До цих процесів можна також віднести діагностику захворювань кімнатних рослин [10], що актуальна і для непрофесіоналів.

У промисловості та машинобудуванні основні способи застосування обробки зображень зосереджені на якості деталей автоматичного обладнання та автоматизованому сортуванні продукції [11]. Також обробка зображень використовується для перевірки якості продукції. Наприклад, під час розливання рідкої продукції або пакування товарів.

В сфері інформаційних технологій цифрова обробка зображень також займає важливе місце. Починаючи, з прямого застосування в програмах з фільтрації і обробки фотографій (мобільні додатки Instagram або Snapchat) [12] та закінчуючи сферою машинного навчання. Алгоритми машинного навчання потребують значної кількості високоякісних даних для тренування та формування правильних результатів роботи. Під час тренування модель аналізує надані зображення у вигляді матриці пікселів. Під час безпосередньо роботи алгоритм звертається до тренувальних даних і виявляє схожість між ними та вхідними даними [13]. Відповідно, для моделей, що працюють з розпізнаванням та класифікацією зображень, крок цифрової обробки зображень є надзвичайно важливим під час формування тренувальних даних моделі, а також під час аналізу вхідних даних.

Припустимо, розробник хоче побудувати алгоритм, який визначить, чи є на зображенні собака чи кіт. Для цього потрібно зібрати зображення собак і котів та попередньо обробити їх.
Етапи попередньої обробки включають:

- Перетворення всіх зображень в однаковий формат та розмір;
- Обрізання непотрібних областей на зображеннях, вибір ключових об’єктів (собак та котів, в даному контексті).

Зазвичай, попередня обробка зображень, тренування та робота моделі машинного навчання відбуваються лише на потужних комп’ютерах. Однак з розвитком можливостей мобільних телефонів, популярність набирає машинне

навчання на пристрої (on-device ML). Відповідно, попередня обробка зображень, як обов'язковий крок процесу класифікації зображень, має також бути перенесена на мобільний пристрій. Правильна обробка зображення може зменшити час навчання моделі та збільшити швидкість її роботи, що є дуже важливим для кінцевого користувача. Наприклад, якщо вхідні зображення занадто великі, зменшення розміру цих зображень може покращити час навчання та роботи моделі без значного зниження її продуктивності. Подібні підходи активно використовуються в системних додатках камери на різних пристроях. До прикладу, стандартний розмір зображень з камери iPhone 11 становить 3024×4032 пікселі. Модель машинного навчання, яку Apple використовує для створення масок і застосування портретного режиму, працює на зображеннях вдвічі меншого розміру. Лише після формування вихідного зображення воно конвертується в повний розмір [14].

1.4 Висновок до розділу

В даному розділі було розглянуто основи цифрової обробки зображень та сформовано термінологічний ряд, що буде використаний в процесі роботи. Також було розглянуто основні сфери застосування цифрової обробки зображень та її роль в них. Особливу увагу було приділено галузі машинного навчання, а зокрема її роботі на мобільних пристроях.

Розділ 2. Інструменти цифрової обробки зображень

З огляду на актуальність процесів обробки зображень та зростаючу потребу у виконанні даних процесів на мобільних пристроях, дана робота фокусується на огляді технологій обробки зображень на пристроях Apple та розширенні наявних інструментів. Це зумовлено тим, що компанія Apple використовує технології машинного навчання та комп'ютерного зору у власних програмах та рекомендує розробникам додавати подібні функції до їхніх додатків. Також активно розробляються інструменти тренування моделей та API з використання готових моделей в застосунках для пристроїв з операційними системами iOS та macOS - фреймворки Core ML, Create ML та Vision.

Під час виконання роботи було вирішено розглянути пристрої з операційною системою iOS та MacOS та доступні для них інструменти цифрової обробки зображень. Дані пристрої включають в себе усі мобільні телефони, планшети та комп'ютери створені компанією Apple. В даному розділі буде розглянуто доступні інструменти для роботи з цифровими зображеннями на пристроях з операційною системою iOS та MacOS.

2.1 Загальні відомості про операційні системи iOS та macOS

iOS (iPhone Operating System) — це мобільна операційна система, що встановлена на мобільних пристроях Apple, таких як iPhone (мобільний телефон) та iPad (планшет). Це також основа для Apple tvOS, яка успадковує багато функцій від батьківської операційної системи. iOS була випущена в червні 2007 року, як операційна система для першої моделі iPhone. iOS є операційною системою на основі Unix, яка працює на всіх мобільних пристроях Apple. iOS — друга за популярністю операційна система для мобільних пристроїв. Станом на березень 2023 року Apple iOS займає 28,42% ринку, поступаючись за поширеністю лише Android, яка займає 70,88% ринку [15].

MacOS — це операційна система Unix, що розробляєтсья компанією Apple з 2001 року. Це основна операційна система для комп'ютерів Mac від Apple.

MacOS є наступником операційної системі Mac, що випускалась з 1984 по 1999 рік. На ринку комп'ютерів це друга за популярністю операційна система після Microsoft Windows. Станом на березень 2023 року Apple macOS займає 17,21% ринку, в той час як основний конкурент, Microsoft Windows, займає 69,4% [16].

Основними інструментами з розробки додатків під операційні системи iOS та macOS є наступні:

- Середовище розробки Xcode. Створене компанією Apple, воно підтримує роботу з декількома мовами програмування, а також дозволяє розробляти та розповсюджувати додатки для iOS та macOS.
- Мова програмування Swift. Високорівнева та мультипарадигменна мова програмуванна розроблена команією Apple, випущена в 2014 році. Swift був розроблений на заміну попередній мові програмування Objective-C, що не оновлювалась та не мала зручних властивостей.

2.2 Інструменти обробки зображень

2.2.1 Фреймворк Core Image

Основним інструментом з цифрової обробки зображень в додатках під iOS та macOS є фреймворк Core Image. Згідно з офіційною документацією [18], Core Image призначений для обробки фото- та відеозображень з найменшими затримками. За допомогою оптимізованих фільтрів та розподілу обчислень можна досягти обробку кадрів отриманих з камери девайсу в реальному часі. Фреймворк приховує низькорівневі деталі обробки зображень, надаючи простий інтерфейс для користування та розширення. Процеси обробки зображень можуть відбуватися як на центральному, так і на графічному процесорі. В загальному випадку, фреймворк автоматично визначає найбільш оптимальний процесор для обчислень, однак за потреби можна вказати його самостійно.

Однією з найважливіших частин фреймворку є фільтри - програми, що призначені для безпосередньої обробки зображень або генерації нових зображень, таких як штрих-коди або дискретні кольори. Core Image має

архітектуру плагіна, містить близько 200 вбудованих фільтрів [18] та можливість розширювати цей набір за допомогою власних фільтрів [19].

Core Image створено для максимального використання апаратного забезпечення, на якому він працює. Комбінації фільтрів оптимізовані таким чином, що код для обробки зображень окремих фільтрів об'єднується разом для створення єдиної програми [20]. Це означає, що там, де це можливо, немає проміжних буферів або зображень, завдяки чому, робота великих наборів фільтрів є дуже швидкою. Процес обчислення не відбувається, поки не вимагається кінцеве зображення, отже система не перевантажується додатково в процесі роботи програми.

Існує два способи створення власних фільтрів для обробки зображень за допомогою фреймворку Core Image:

- Комбінування існуючих фільтрів;
- Створення нових фільтрів на основі власних ядер (kernels).

Ядро фільтру (kernel) представляє собою програму, що виконує обробку кожного пікселя зображення. Усі вбудовані фільтри Core Image містять принаймні одне ядро, деякі можуть містити два чи більше. Фреймворк Core Image надає три типи ядер: кольорові (color kernel) [21], деформуючі (warp kernel) [22] та загальні (general kernel) [23].

Кольорові ядра оптимізовано для зміни кольору окремих пікселів, при цьому вони не містять жодної інформації про координати даного пікселя. Прикладом фільтру, що базується на кольоровому ядрі є фільтр, що змінює експозицію зображення.

Ядра деформації оптимізовані для зміни геометрії зображення, тому в процесі обробки вони не володіють інформацією про колір пікселя. Наприклад, масштабування та обрізка зображення.

Загальні ядра можуть брати вибірку кількох пікселів у зображенні, а також змінювати колір і розміщення даного пікселя. Вони використовуються в таких фільтрах, як розмиття та підвищення різкості.

Ядра загального типу можуть обраховувати значення поточного пікселя в залежності від інших пікселів зображення, однак коли в цьому немає потреби, варто застосовувати ядро визначеного типу, що дозволить Core Image забезпечити кращу продуктивність фільтра.

2.2.2 Custom kernels

Власні ядра (будь-якого типу) можуть бути написані мовою програмування Core Image Kernel або мовою Metal Shading (для пристроїв з процесором A8 та новішими). Обидві ці мови є шейдерними. Шейдерна мова — це мова графічного програмування, що адаптована для програмування зображень, текстур та поверхонь.

Програма, написана мовою Core Image Kernel Language, повинна містити одну або кілька процедур обробки зображень у рядковому форматі (String). Вихідний код аналізується в процесі створення об'єкту класу CIKernel бібліотеки Core Image. Під час безпосередньої обробки та візуалізації зображень Core Image може об'єднувати різні ядра та створювати оптимізовані шейдерні програми [24].

Metal Shading Language також може використовуватися для створення шейдерних програм та їх інтеграції в процес обробки зображень за допомогою Core Image. Це мова програмування в стилі C++14, яка зазвичай використовується для написання програм, що призначені для роботи на графічному процесорі (GPU) [25].

Основними перевагами написання ядер мовою Metal Shading Language є:

- Попередня компіляція під час збірки (програма мовою Core Image Kernel Language компілюється під час виконання);
- Підтримка комбінування програм та тайл шейдингу: рендеринг окремих частин зображення та збереження результатів в часткову пам'ять (tile memory) на графічному процесорі;
- Оновлені властивості та більш сучасний синтаксис;

- Можливість поєднання з іншими програмами ядер, що не написані даною мовою.

З огляду на перераховані переваги, Metal Shading Language є найбільш оптимальною та зручною мовою для написання ядер - основи фільтрів цифрової обробки зобрежень в середовищі Apple. Після розробки ядра, на основі нього можна створити власний фільтр та інтегрувати його в пайплайн обробки зображень в програмі.

Як ілюстрацію роботи з Metal Shading Language, розглянемо процес створення ядра для перетворення пікселів зображення на ті, що містять тільки відтінки сірого. Такі зображення часто є більш зручними для подільшої обробки, адже займають менше виділеної пам'яті. Існує ряд поширених методів перетворення кольорових зображень у монохромні, ті, що містять тільки відтінки сірого, наприклад метод середнього значення та зважений метод [26].

Метод середнього значення обчислює значення пікселя як середнє значень його кольорових каналів (червоного, зеленого та синього).

$$Y' = \frac{(R' + G' + B')}{3} \qquad (2.1)$$

Дана формула *(2.1)* є коректним математичним представленням, однак під час написання коду, що описує її, можна зіштовхнутися з проблемою переповнення змінної, що позначає суму значень. Сума значень трьох кольорових каналів може перевищувати максимальне числове значення, що використовуваний тип змінних (*uint8*) може вміщувати - 255. Щоб уникнути цієї помилки, значення варто обчислювати окремо, за формулою *(2.2)*.

$$Y' = \frac{R'}{3} + \frac{G'}{3} + \frac{B'}{3} \qquad (2.2)$$

Метод середнього значення є простим для розуміння, але не надає повністю коректного вихідного значення. Це зумовлено тим, що людське око по-

різному реагує на кольори спектру. Очі людини найбільш чутливі до зеленого світла, середньо чутливі до червоного світла і найменш чутливі до синього світла [27]. Тому три кольори повинні мати різну вагу під час обчислення. Базуючись на цьому побудований зважений метод, або метод освітленості.

Зважений метод розподіляє червоний, зелений і синій кольори відповідно до довжин їхніх хвиль.

$$Y' = 0.299R' + 0.587G' + 0.114B' \qquad (2.3)$$

Програма, що виконує дане перетворення для кожного пікселя зображення виглядає наступним чином *(Рисунок 2.1)*. На даному прикладі можна розібрати ключові типи змінних та структуру програми-ядра.

```
float4 grayscaleKernel(sample_t s) {
    float gray = (0.2989 * s.r + 0.5870 * s.g + 0.1140 * s.b);
    return float4(gray, gray, gray, s.a);
}
```

Рисунок 2.1 — Програма-ядро з перетворення пікселів зображення

Вхідний параметр типу *sample_t*. Тип *sample_t* представляє собою один піксель із вхідного зображення, оскільки ядро застосовується до кожного пікселя. *sample_t* — це 4-вектор чисел з плаваючою комою, сформований за типом RGBA - канали червоного, зеленого та синього кольорів, а також значення прозорості. Використовується як тип параметра лише для представлення пікселів зображення. Інакше поводиться як звичайний 4-вектор - *float4*. Іншим можливим типом вхідного параметру є *sampler*, що містить дані про ціле зображення. Вихідний параметр типу *float4*. float4 є вектором тієї ж структури, що й *sample_t*, та представляє значення кожного каналу вихідного пікселя.

2.2.3 Custom Filters

Незалежно від мови та способу написання ядра фільтра, його треба інтегрувати у власний клас, що наслідує клас CIFilter. Для цього фреймворк Core Image надає програмний інтерфейс, розробка відбувається мовою програмування Swift.

Створення власного фільтра складається з наступних кроків [28]:

- Оголошення вхідних параметрів фільтра. Рекомендовано також додавати до імені вхідного параметра префікс input, наприклад inputImage.
- Ініціалізація ядра певного типу з програмного файлу;
- Перевизначення вихідного параметру - outputImage - із застосуванням ядра обробки до вхідного зображення.

Отриманий фільтр можна застосовувати до будь-якого зображення відповідного типу. За потреби його також можна зареєструвати серед фільтрів Core Image, щоб додати можливість ініціалізувати фільтр за допомогою його назви. Варто зауважити, що не можна використовувати назви вже існуючих фільтрів для реєстрації власних.

Таким чином, бібліотека Core Image дозволяє використовувати вбудовані фільтри, об'єднувати їх в набори, а також створювати власні фільтри будь-якої складності. За допомогою наданих інструментів можна створювати різноманітні проекти з обробки зображень або інтегрувати роботу із зображеннями в будь-які проекти.

2.3 Висновок до розділу

В даному розділі було розглянуто наявні інструменти для роботи з цифровими зображеннями на пристроях з операційною системою iOS та MacOS. Було визначено основні фреймворки та мови програмування, що можуть бути використані для імплементації процесів цифрової обробки зображень, та інтеграції цих процесів в додатки написані мовою програмування Swift.

Розділ 3. Розробка власної бібліотеки

3.1 Вимоги та інструменти для розробки і розповсюдження бібліотеки

За мету даної роботи було взято зокрема створення фреймворку для обробки зображень на пристроях з операційною системою iOS та macOS. Даний фреймворк має містити допоміжні функції для розробників та поділятися на три основних розділи: службові функції (зчитування та відображення зображень); згорткові фільтри; програми-генератори (шум Вороного, шум Перліна тощо). Фреймворк не призначений для заміни наявних інструментів, навпаки, для їх доповнення та розширення. Існуючі бібліотеки, хоч і містять вбудовані фільтри, не покривають усіх потрібних методів обробки та генерації зображень, що можуть бути використані в базових процесах. Таким чином, фреймворк може задовольнити додаткові потреби розробників та спростити роботу з обробкою зображень на пристроях.

В основу фреймворку було покладено наступні бібліотеки: CoreImage, MetalKit.

Бібліотека CoreImage – це основний інструмент з цифрової обробки зображень в додатках під iOS та macOS. Класи даного фреймворку використано як основу для фільтрів та генераторів зображень.

MetalKit – це фреймворк для рендерингу графічного контенту за допомогою графічного процесора. Основні функції даної бібліотеки дають можливість відображати оброблені зображення значено швидше.

Під час розробки не було використано жодної системної бібліотеки, що була б специфічною для однієї з операційних систем (iOS або macOS), що дає можливість використовувати функції фреймворку незалежно від системи та пристрою.

Розробка відбувалася двома мовами програмування. Мова Swift була використана для створення безпосередньо фільтрів та розширень бібліотеки Core Image, а також для розробки допоміжних елементів для розробника. Мова Metal Shading Language була викристана для розробки ядер (kernel) власних фільтрів

та генераторів. В якості середовища розробки було обрано Xcode - основне та найбільш зручне середовище розробки для проектів мовою Swift.

Фреймворк було зібрано у формат (.framework), придатний для додавання в програмні проекти, та розповсюджено за допомогою платформи Github. Під час аналізу доступних інструментів для розповсюдження було розглянуто також менеджери залежностей, однак вони не задовольнили потреби проекту.

Менеджер залежностей — це програмне забезпечення, що спрощує процес встановлення, оновлення, налаштування та видалення стороннього програмного забезпечення у власних проектах. Найбільш поширеними менеджерами залежностей для фреймворків під операційні системи iOS/macOS є Swift Package Manager та Cocoapods. Обидва ці менеджери автоматично збирають фреймворки та надають можливість розробникам використовувати їх у своїх проектах вже у вигляді готових елементів. Відповідно, розробник не може впливати на процес збірки. Однак такі фреймворки, що містять фільтри обробки зображень із ядрами, написаними мовою Metal Shading Language, потребують використання специфічних параметрів під час збірки. Таким чином, фреймворк з обробки зображень із власними фільтрами не може бути розповсюдженим за допомогою жодного з менеджерів залежностей.

3.2 Розробка службових функцій

Під час розробки допоміжних функцій було вирішено сфокусуватися на спрощенні та прискоренні двох основних процесів роботи з зображеннями: отримання зображення з камери та відображення обробленого зображення на екрані пристрою.

Для отримання зображення з камери пристрою було використано системну бібліотеку AVFoundation, як основу для обгортки. Вона надає розробниками функції для створення сесії зчитування зображень покадрово. Надалі ці кадри можна конвертувати в придатний для обробки формат, що значним чином спрощує роботу.

Для рендерингу та відображення зображення було використано системну бібліотеку MetalKit. На основі класів цієї бібліотеки було створено допоміжні класи, що вміщують роботу з графічним процесором, рендеринг зображення та власне його відображення. Робота з MetalKit щільно пов'язана із графічним процесором пристрою: розробник надсилає команди для рендерингу, процесор виконує необхідні задачі та повідомляє про результат роботи, використовуючи патерн делегат. Такий спосіб дозволяє досягти значного пришвидшення в процесі відображення проміжних або кінцевих зображень.

3.3 Згорткові фільтри

Згорткові фільтри (convolution filters) використовуються для розмиття зображень, підвищення різкості, виявлення країв об'єктів тощо. Це досягається процесом згортки між згортковою матрицею та зображенням. Матриці зазвичай мають розміри 3x3, однак можуть і відрізнятися за розміром, залежно від потреби. Процес згортки можна виразити математично наступним чином (за формулою *(3.1)*). Варто зауважити, що згортка не є звичайним множенням матриць, хоч і позначається тим же символом.

$$g(x,y) = \omega * f(x,y) = \sum_{dx=-a}^{a} \sum_{dy=-b}^{b} \omega(dx,dy) f(x+dx, y+dy) \quad (3.1)$$

де g(x,y) — вихідне відфільтроване зображення;
f(x,y) — початкове зображення;
w — матриця згортки фільтра.

Розглянемо застосування згортки до пікселя зображення з координатами (1, 1):

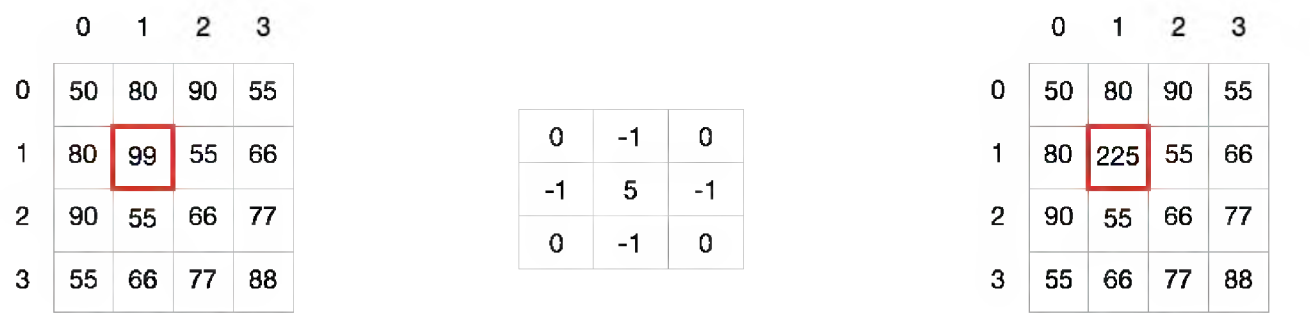

Рисунок 3.1 — Застосування згортки до певного пікселя зображення

Значення будь-якого пікселя у вихідному зображенні обчислюються шляхом множення кожного значення згорткової матриці на відповідні значення пікселів вхідного зображення. Цей процес виконується для всіх пікселів зображення, окрім граничних. Є кілька способів обробки граничих пікселів, серед яких найпоширенішими є розширення, обрізання та використання констант.

Найбільш відомими згортковими фільтрами є підвищення різкості, визначення країв та розмиття. Для кожного з цих фільтрів є певна згорткова матриця, яку треба застосувати до вхідного зображення.

Для роботи із квадратними згортковими матрицями бібліотека Core Image пропонує три вбудованих фільтри: convolution3X3Filter, convolution5X5Filter, convolution7X7Filter. Дані фільтри приймають зображення, а також значення матриці певного розміру для згортки. Однак робота зі згортковими фільтрами зазвичай вимагає використання вже існуючих фільтрів, а не створення власних. Таким чином, розробник може потребувати фільтр розмиття Гауса (один із найпоширеніших згорткових фільтрів), а не просто загальний фільтр, що потребує додатково пошуку потрібної матриці.

Таким чином, було вибрано 29 найбільш використовуваних згорткових фільтрів, та додано їх у розроблену бібліотеку. Дані фільтри базуються на сталих

згорткових матрицях та фільтрах наданих бібліотекою Core Image, що спрощує їхнє використання у власних проектах.

3.4 Генератори текстур та шумів

У комп'ютерній графіці процедурна текстура— це текстура, що створена з використанням математичного алгоритму, а не безпосередньо із цифрових зображень [29]. Перевагою цього підходу є необмежена роздільна здатність текстури та просте відображення. Такі типи текстур часто використовуються для моделювання поверхонь або об'ємних зображень природних елементів, таких як дерево, мармур, граніт, метал, камінь та інші. Такі текстури можуть бути використані у розробці ігор, генерації 3D зображень та 2D зображень високої якості.

Шум - це примітивний елемент процедурної текстури. Це випадковий і неструктурований шаблон, і він корисний у тих випадках, коли є потреба в джерелі великої кількості деталей, яким, однак, бракує очевидної структури [30]. Є два основних типи шуму: знаковий шум (value noise) та градієнтний шум (gradient noise).

Знаковий шум використовує псевдовипадкове число у якості вхідного значення, а нахил функції в кожній точці інтервалу дорівнює нулю.

Градієнтний шум використовує кожну точку інтервалу як вхідне значення, а псведовипадкове число отримане в цій точці використовуються як нахил (градієнт) кривої.

Під час розробки бібліотеки було обрано найпоширеніші приклади процедурних текстур та шумів, таких як шум Вороного (Ворлі), шум Перліна та Симплекс шум, та реалізовано їх за допомогою мови Metal Shading Language. Було створено функції генерації даних структур, що в результаті дає можливість напряму використовувати структури або створювати нові елементи на їх основі.

Висновки

Результатом даної роботи є фреймворк з обробки зображень для використання у програмах, написаних мовою Swift. Даний фреймворк розширює можливості вже існуючих інструментів та надає розробникам доступ до великої кількості функцій з цифрової обробки зображень. Фреймворк реалізує алгоритми обробки зображень мовою програмування Swift та містить допоміжний код мовою Metal Shading Language для роботи з графічним процесором пристрою. Фреймворк розповсюджено у вигляді готового пакету з відкритим кодом.

Під час виконання даної роботи було проведено комплексний огляд методів цифрової обробки зображень, їхніх типів та основних складових. Було розглянуто основи обробки зображень та визначено сфери її застосування.

Було проаналізовано інструменти з обробки зображень на пристроях з операційними системами iOS та macOS. Було проведено детальний огляд архітектури та функцій доступних фреймворків, а також приклади їх використання для завдань обробки зображень. Було досліджено використання Core Image, високорівневого фреймворку для обробки зображень, та мови програмування Metal Shading Language, для прискорення завдань обробки зображень на пристроях Apple.

Список використаної літератури

1. C. Luo, Y. Hao, and Z. Tong, “Research on Digital Image Processing Technology and Its Application,” Proceedings of the 2018 8th International Conference on Management, Education and Information (MEICI 2018). Atlantis Press, 2018. doi: 10.2991/meici-18.2018.116.

2. A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, “Medical image processing on the GPU – Past, present and future,” Medical Image Analysis, vol. 17, no. 8. Elsevier BV, pp. 1073–1094, Dec. 2013. doi: 10.1016/j.media.2013.05.008.

3. A. Kamalakannan and G. Rajamanickam, “High Performance Color Image Processing in Multicore CPU using MFC Multithreading,” International Journal of Advanced Computer Science and Applications, vol. 4, no. 12. The Science and Information Organization, 2013. doi: 10.14569/ijacsa.2013.041207.

4. Y. Huang, “Overview of Research Progress of Digital Image Processing Technology,” Journal of Physics: Conference Series, vol. 2386, no. 1. IOP Publishing, p. 012034, Dec. 01, 2022. doi: 10.1088/1742-6596/2386/1/012034.

5. R. Gonzalez, Digital Image Processing. Third Edition, 2008. – 976 с.

6. M. Sarfraz, “Introductory Chapter: On Digital Image Processing,” Digital Imaging. IntechOpen, May 13, 2020. doi: 10.5772/intechopen.92060.

7. H. Kruegle, CCTV Surveillance. Analog and Digital Video Practices and Technology, Second Edition, 2006. – 673 с.

8. C. Pirazzi, "Programmers' Guide to Video Systems” [Електронний ресурс], 2008 - Режим доступу до ресурсу: https://lurkertech.com/lg/pixelaspect/.

9. N. Koren, “Making fine prints in your digital darkroom. Pixels, images, and files” [Електронний ресурс], 2004 - Режим доступу до ресурсу: http://www.normankoren.com/pixels_images.html

10. J. G. Arnal Barbedo, “Digital image processing techniques for detecting, quantifying and classifying plant diseases,” SpringerPlus, vol. 2, no. 1. Springer Science and Business Media LLC, Dec. 2013. doi: 10.1186/2193-1801-2-660.

11. R. A. Ciora and C. M. Simion, “Industrial Applications of Image Processing,” ACTA Universitatis Cibiniensis, vol. 64, no. 1. Walter de Gruyter GmbH, pp. 17–21, Nov. 01, 2014. doi: 10.2478/aucts-2014-0004.

12. A. Ahmad, “Snapchat’s Filters: How Computer Vision Recognizes Your Face”, [Електронний ресурс], 2018 - Режим доступу до ресурсу: https://www.evolvear.io/augmented-reality-blog/snapchats-filters-how-computer-vision-recognizes-your-face/

13. A. Goswami et al., Change Detection in Remote Sensing Image Data Comparing Algebraic and Machine Learning Methods. Electronics, 2022, https://doi.org/10.3390/electronics11030431

14. J. Nelson, “What is Image Preprocessing and Augmentation?” [Електронний ресурс], 2020 - Режим доступу до ресурсу: https://blog.roboflow.com/why-preprocess-augment/

15. Mobile Operating System Market Share Worldwide [Електронний ресурс], 2023 - Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/mobile/worldwide

16. Desktop Operating System Market Share Worldwide [Електронний ресурс], 2023 - Режим доступу до ресурсу: https://gs.statcounter.com/os-market-share/desktop/worldwide

17. Core Image Programming Guide [Електронний ресурс], 2016 - Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/GraphicsImaging/Conceptual/CoreImaging/ci_intro/ci_intro.html

18. Core Image Filter Reference [Електронний ресурс], 2016 - Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/GraphicsImaging/Reference/CoreImageFilterReference/index.html

19. Core Image Filter Reference, What You Need to Know Before Writing a Custom Filter [Електронний ресурс], 2016 - Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/GraphicsImaging/Concept

ual/CoreImaging/ci_advanced_concepts/ci.advanced_concepts.html#//apple_ref/doc/uid/TP30001185-CH9-SW1

20. CIImageProcessorKernel [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/documentation/coreimage/ciimageprocessorkernel

21. CIColorKernel [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/documentation/coreimage/cicolorkernel

22. CIWarpKernel [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/documentation/coreimage/ciwarpkernel

23. CIKernel [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/documentation/coreimage/cikernel

24. Core Image Kernel Language Reference [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/metal/CoreImageKernelLanguageReference11.pdf

25. Metal Shading Language Specification [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/metal/Metal-Shading-Language-Specification.pdf

26. C. Kanan and G. W. Cottrell, “Color-to-Grayscale: Does the Method Matter in Image Recognition?,” PLoS ONE, vol. 7, no. 1. Public Library of Science (PLoS), p. e29740, Jan. 10, 2012. doi: 10.1371/journal.pone.0029740.

27. A. Grzybowski and K. Kupidura-Majewski, “What is color and how it is perceived?,” Clinics in Dermatology, vol. 37, no. 5. Elsevier BV, pp. 392–401, Sep. 2019. doi: 10.1016/j.clindermatol.2019.07.008.

28. Core Image Programming Guide, Creating Custom Filters [Електронний ресурс] - Режим доступу до ресурсу: https://developer.apple.com/library/archive/documentation/GraphicsImaging/Conceptual/CoreImaging/ci_custom_filters/ci_custom_filters.html

29. PCMag Encyclopedia, procedural texture [Електронний ресурс] - Режим доступу до ресурсу: https://www.pcmag.com/encyclopedia/term/procedural-texture

30. A. Lagae et al., “A Survey of Procedural Noise Functions,” Computer Graphics Forum, vol. 29, no. 8. Wiley, pp. 2579–2600, Oct. 04, 2010. doi: 10.1111/j.1467-8659.2010.01827.x.