Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматики
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі»

Виконав: студент 2-го курсу групи 1ІСТ-23м
спеціальності 126 – Інформаційні системи та технології
Владислав ШМАЛЮХ

Керівник: к.т.н., доц. каф. АІІТ
Ілона БОГАЧ
«16» грудня 2024 р.

Опонент: к.т.н., доц. каф. КН
Людмила КРИЛИК
«16» грудня 2024 р.

Допущено до захисту
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«17» грудня 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра автоматизації та інтелектуальних інформаційних технологій

Рівень вищої освіти II-ий (магістерський)

Галузь знань – 12 – Інформаційні технології

Спеціальність – 126 Інформаційні системи та технології

Освітня програма – Інформаційні технології аналізу даних та зображень

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«19» 09 2024 р.

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Шмалюху Владиславу Анатолійовичу

1. Тема роботи: Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі
2. Керівник роботи: к.т.н., доцент каф. АІІТ Ілона БОГАЧ

Затверджені наказом ВНТУ від «17» 09 2024 року №310.

3. Строк подання студентом роботи до «16» 12 2024 року.
4. Вихідні дані до роботи: операційна система Ubuntu 20.04; мова програмування Java, тип піксела – гексагон, розмір координатного простору – 4096*4096; тип інтерполяції – лінійна, кола; тип антиаліайзингу – контурний; розмір піксельного простору між сусідніми пікселями – 1 дискрета, середовище розробки – IntelliJ IDEA; вихідні дані: сформоване зображення на гексагональному растрі.
5. Зміст текстової частини: вступ; аналіз об'єкту досліджень; розробка алгоритмів і математичних моделей для роботи з шестикутним растром; проєктування інформаційної технології для оптимізації зображень; проектування та реалізація модуля конвертації зображень; економічна частина; висновки; список використаних джерел.
6. Перелік ілюстративного (або графічного) матеріалу: схема алгоритму формування базових графічних примітивів; схема потоку даних для конвертації зображень; схема роботи антиаліайзингу на шестикутному растрі; схема архітектури програмного модуля конвертації; схема активності побудови складних зображень; схема узгодження та взаємодії компонентів модулю конвертації.

Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ім'я та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Ілона БОГАЧ, к.т.н., доцент каф. АІІТ	11.09.24	16.12.24
5	Олександр ЛЕСЬКО, к.е.н., проф. каф. ЕПтаВМ	10.09.24	10.12.24

7. Дата видачі завдання «11» 09 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи		Примітка
1	Аналіз предметної області та системи, що слід оптимізувати	16.09.24	22.09.24	виконано
2	Вибір інструментів розробки	23.09.24	09.10.24	виконано
3	Проектування архітектури та розробка компонентів системи	10.10.24	30.10.24	виконано
4	Аналіз та оптимізація системи	01.11.24	05.11.24	виконано.
5	Економічна частина	06.11.24	20.11.24	виконано.
6	Оформлення пояснювальної записки і графічного матеріалу	20.11.24	30.11.24	виконано.
7	Попередній захист роботи	01.12.24	05.12.24	виконано.
8	Остаточний захист роботи	19.12.24		

Студент _____ Владислав ШМАЛЮХ
(підпис)

Керівник роботи _____ Ілона БОГАЧ
(підпис)

АНОТАЦІЯ

УДК 004.92

Шмалюх В. А. Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі. Магістерська кваліфікаційна робота зі спеціальності 126 – Інформаційні системи та технології, освітня програма – Інформаційні технології аналізу даних та зображень. Вінниця: ВНТУ, 2024. 160с.

На укр. мові. Бібліогр.: 40 назви; рис.: 62; табл.: 20.

У роботі розроблено інформаційну технологію для оптимізації побудови зображень на шестикутному растрі. Представлено математичні моделі та програмні рішення для перетворення растрових зображень у шестикутний формат, розроблено методи побудови базових графічних примітивів і реалізовано засоби покращення продуктивності. Проведено аналіз ефективності розроблених алгоритмів, виконано порівняння з аналогами за технічними та економічними показниками.

Ключові слова: шестикутний растр, графічні примітиви, оптимізація зображень, алгоритми побудови, математичне моделювання, інформаційні технології, візуалізація даних.

ABSTRACT

Vladyslav SHMALIUKH. Information Technology for Optimizing Image Construction on a Hexagonal Grid. Master's qualification paper of a specialty 126 – Informational systems and technologies, curricula – Informational technologies of data and image analysis. Vinnytsia: VNTU, 2024. 160 pages.

In Ukrainian. Bibliography: 40 sources; figures: 62; tables: 20.

This thesis develops information technology for optimizing image construction on a hexagonal grid. Mathematical models and software solutions for converting raster images into a hexagonal format are presented. Methods for constructing basic graphical primitives are developed, and tools for performance optimization are implemented. The efficiency of the developed algorithms is analyzed, and comparisons are made with analogs based on technical and economic indicators.

Keywords: hexagonal grid, graphical primitives, image optimization, construction algorithms, mathematical modeling, information technology, data visualization.

ЗМІСТ

ВСТУП

Актуальність роботи. Шестикутні растри є перспективним напрямом у комп'ютерній графіці та обробці зображень завдяки своїм унікальним властивостям, таким як підвищена щільність упаковки пікселів, симетричність і зменшення артефактів при інтерполяції. Ці особливості роблять гексагональні структури придатними для розробки високоякісних графічних алгоритмів і програмних рішень, що можуть бути використані в різних галузях.

Серед прикладних сфер, де гексагональні растри мають значний потенціал, можна виділити геоінформаційні системи, системи моделювання природних процесів, комп’ютерне бачення, а також створення високореалістичних графічних ефектів. Використання такої растрової структури дозволяє отримати якісніші результати візуалізації та підвищити продуктивність алгоритмів.

Проте впровадження гексагональних растрових структур вимагає розробки нових методів і алгоритмів, адаптованих до специфічних властивостей цього формату. Зокрема, це стосується процесів формування базових графічних примітивів, таких як відрізки прямих, кола, та застосування методів усунення аліайзингу, які мають суттєвий вплив на якість зображень. Таким чином, дослідження в цьому напрямі є актуальним і важливим як у теоретичному, так і в практичному аспектах.

Обраний напрям дослідження також відповідає сучасним тенденціям розвитку інформаційних технологій, зокрема інтеграції математичних моделей, алгоритмів обробки зображень та програмного забезпечення. Це забезпечує комплексний підхід до розв’язання завдань оптимізації формування зображень, що має значення для розширення наукового потенціалу та практичного використання в інформаційних системах.

Актуальністю роботи є необхідність вирішення актуальних завдань, пов’язаних із підвищенням реалістичності, якості та продуктивності формування зображень на гексагональному растрі.

Мета роботи полягає в розширенні функціональних можливостей інформаційної технології для оптимізації процесу формування зображень на гексагональному растрі шляхом імплементації нових математичних моделей, алгоритмів і програмних засобів, які забезпечують високу якість візуалізації, підвищення продуктивності обчислень та зниження кількості артефактів.

Для досягнення поставленої мети необхідно вирішити такі завдання:

1. Провести аналіз існуючих методів формування зображень та графічних примітивів на гексагональних растрах, визначити їх переваги та недоліки.

2. Розробити математичні моделі та алгоритми для формування базових графічних примітивів (ліній, кіл, полігонів) на гексагональному растрі.

3. Запропонувати та реалізувати методи оптимізації продуктивності формування примітивів шляхом використання інтерполяційних і аналітичних підходів.

4. Розробити алгоритм конвертації стандартних растрових зображень у гексагональний формат, забезпечуючи збереження якості та деталізації.

5. Реалізувати методи усунення артефактів, зокрема застосування алгоритмів антиліайзингу для підвищення якості відображення.

6. Створити програмний модуль, що інтегрує розроблені алгоритми, з урахуванням зручності використання для кінцевого користувача та виконати його тестування.

Об’єкт дослідження – процес формування зображень і графічних примітивів на шестикутному растрі.

Предмет дослідження – методи, алгоритми та програмні засоби для формування зображень і графічних примітивів на гексагональному растрі.

Методи дослідження. У процесі дослідження застосовувалися: теорія чисел і чисельних методів; теорія інтерполювання функцій; методи аналітичної геометрії для розробки методів і засобів формування графічних примітивів на гексагональному растрі; комп’ютерне моделювання для аналізу формування примітивів на гексагональному растрі.

Науково-практичний результат роботи:

Удосконалено методи та алгоритми, що дозволяють ефективно реалізувати формування графічних примітивів на гексагональному растрі, які раніше не знаходили широкого використання через складність адаптації до специфічної структури і які забезпечують високу якість візуалізації, підвищення продуктивності обчислень та зниження кількості артефактів.

Практична цінність отриманих результатів: створено програмний модуль для конвертації растрових зображень у формат гексагонального растру, що відкриває нові можливості для інтеграції технології в сучасні інформаційні системи.

Особистий внесок здобувача. Усі наукові результати отримано здобувачем самостійно. У працях, опублікованих у співавторстві, студенту належать: [2] – розроблено граф схему для імітації гексагонального растру; [3] – виконано аналіз особливості інтегрованої графіки; [4] – виконано аналіз методів тестування програмних застосунків; [5] – виконано огляд програм для аналізу даних; [6] – проаналізовано алгоритм усунення аліайзингу; [7] – розроблено алгоритм для тренування операторів БПЛА; [8] – проаналізовано алгоритм підвищення яскравості екранів; [9] – наведено метод підвищення продуктивності комп'ютерних ігор; [10] – проаналізовано стандартів створення програмних продуктів; [11] – проведено аналіз особливостей стиснення зображення; [12] – проаналізовано технології паралельної роботи відеокарт; [13] – досліджено технологію для організації процесів розробки програмного забезпечення; [14] – проведено аналіз піксельних шейдерів; [15] – проаналізовано нововведення особливостей RNDA, [16] – розроблено застосунок для формування відрізків на шестикутному растрі, [17] – розроблено алгоритм і виконано обрахунки вагів приростів, [18] – імплементовано алгоритми в програмному коді, [19] – сформовано особливість підходу, [20] – розроблено програмні компоненти.

Апробація результатів роботи. Результати роботи доповідалися на конференціях: II Міжнародна науково-практична конференція «Fundamental and applied research in the modern world» (Бостон, 2020), Міжнародна науково-

практична конференція «Scientific achievements of modern society» (Ліверпуль, 2020), XIX International Scientific and Practical Conference (Франкфурт, 2020), XLIX Науково-технічна конференція Інституту соціально-гуманітарних наук (Вінниця, 2020), Інновації XXI століття, L Міжнародна науково-практична інтернетконференція (Вінниця, 2020), IX Міжнародна науково-практична конференції студентів, аспірантів та молодих вчених «Використання інформаційних та комунікаційних технологій в сучасному цифровому суспільстві» (Харків, 2020), V міжнародна науково-практична конференція «Інформаційні технології в культурі, мистецтві, освіті, науці, економіці та бізнесі» (Київ, 2020), Науково-технічна конференція Інституту соціально-гуманітарних наук (Вінниця, 2021), I всеукраїнська науково-технічної конференції молодих вчених, аспірантів та студентів «Комп'ютерні ігри і мультимедіа як інноваційний підхід до комунікації» (Черкаси, 2021), Конференція молодих вчених, студентів та аспірантів (Conferences of Young Scientists, Students and Postgraduate Students) «Стан, досягнення та перспективи інформаційних систем і технологій» (Одеса, 2021), Всеукраїнська науково-практичної конференції «Нові інформаційні технології управління бізнесом» (2021), XII Міжнародна науково-технічної конференції «Інформаційно-комп’ютерні технології» (Житомир, 2021), LI Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії (Вінниця, 2022), XII Міжнародна науково-практична конференція «MODERN RESEARCH IN WORLD SCIENCE» (Львів, 2023), Міжнародна науково-практична інтернет-конференція «Youth in Science: Research, Problems, Prospects» (Вінниця, 2024).

Публікації. Основні результати досліджень опубліковано в 19 наукових працях у матеріалах республіканських і міжнародних конференціях.

1 АНАЛІЗ ОБ’ЄКТУ ДОСЛІДЖЕНЬ

1.1 Характеристика процесу формування зображень і графічних примітивів на шестикутному растрі

Шестикутний растр є альтернативою традиційним квадратним растрам у комп'ютерній графіці, який пропонує низку переваг завдяки унікальній структурі розташування пікселів. Основною особливістю шестикутного растру є його геометрична організація, що базується на комірках у формі правильних шестикутників. Це забезпечує більш однорідне покриття площини та мінімізує артефакти при відображенні діагональних і криволінійних контурів. У порівнянні з квадратним растром, шестикутний растр дозволяє створювати реалістичніші та візуально більш плавні графічні примітиви [1], [2], [22].

Дослідження шестикутного растру відіграє важливу роль у розвитку програмних засобів у різних сферах [23].

У сфері графічного дизайну шестикутна сітка може стати базою для створення графічних компонентів, таких як значки, логотипи, ілюстрації та інше. Завдяки своїй геометричній симетрії, вона дозволяє з легкістю генерувати різноманітні форми та структури з високою точністю.

В обробці зображень і аналізі комп'ютерного зорового сприйняття шестикутний растр може використовуватися для покращення обробки зображень. Його структура сприяє зниженню спотворень, які можуть виникати під час обробки, та підвищенню точності при аналізі, виконаному системами комп’ютерного зору.

У комп’ютерних іграх шестикутний растр знайшов своє застосування в розробці графічних компонентів, особливо для стратегічних ігор. Цей формат забезпечує реалістичне представлення географічних карт із високою деталізацією та природним виглядом. Крім того, він широко використовується для створення тайлів, що покращує рух об’єктів та їхню взаємодію в ігровому середовищі.

Квадратна сітка, яка є стандартною для більшості моніторів, використовує прямокутні пікселі, розташовані в регулярному порядку. У порівнянні з шестикутною сіткою, вона має низку недоліків. Наприклад, через прямокутну форму пікселів часто виникають викривлення нахилених ліній і геометричних об’єктів, особливо кривих. Це може призводити до ефектів аліайзингу та втрати деталей, особливо на краях криволінійних об'єктів.

Шестикутний растр, у свою чергу, долає ці недоліки. Завдяки своїй геометричній структурі, він забезпечує більш точне відображення нахилених ліній і криволінійних елементів, мінімізуючи ефекти аліайзингу та зберігаючи більше деталей на краях кривих об’єктів. Це робить його перспективним для використання у багатьох галузях комп’ютерної графіки та обробки зображень.

У процесі створення застосунку для роботи з шестикутним растром необхідно забезпечити реалізацію кількох ключових функціональних можливостей, які забезпечать ефективну взаємодію із гексагональною сіткою та підтримку створення зображень [24].

Перша функція полягає у генерації самої гексагональної решітки. Це передбачає розробку методів розрахунку розмірів і координат кожного елемента сітки, а також забезпечення їх коректного розташування й взаємодії між собою.

Друга функція спрямована на створення базових графічних примітивів, таких як точки, відрізки та окружності, на основі гексагональної структури. Це вимагає адаптації алгоритмів для розміщення цих примітивів з урахуванням специфіки шестикутного растру, а також забезпечення їх точної інтеграції в сітку.

Третя функціональна можливість стосується побудови складніших зображень на основі примітивів, розміщених на шестикутному растрі. Це включає комбінування елементів, використання кольорової палітри, накладення текстур і виконання таких операцій, як масштабування, фільтрація та зміна характеристик окремих елементів сітки.

Для забезпечення більшої варіативності та зручності у використанні застосунка передбачається розробка методів налаштування параметрів генерації

зображень. Це може включати вибір алгоритмів формування, встановлення параметрів роботи з примітивами, а також можливість інтерактивно змінювати налаштування програми для досягнення бажаного результату. Завдяки таким рішенням забезпечується адаптивність застосунка до потреб користувачів і підвищується якість кінцевого результату.

Введення таких функціональних можливостей дозволить користувачам ефективно працювати з гексагональним растром і створювати різноманітні зображення, надаючи їм можливість контролювати вигляд і деталі створених об'єктів [25].

1.2 Порівняльний аналіз аналогів

Аналіз існуючих рішень для роботи з гексагональним растром є важливим етапом дослідження, оскільки дозволяє оцінити переваги й недоліки популярних програмних продуктів, які вже використовуються для вирішення подібних завдань [26]. Це обґрунтовує доцільність розробки власного рішення, враховуючи його специфіку та потреби користувачів. Основними критеріями для аналізу обрано функціонал, зручність у використанні, точність формування зображень, вартість програмного продукту, підтримку роботи з гексагональним растром та можливість налаштування параметрів. Ці характеристики дозволяють повноцінно оцінити ефективність і придатність програмного забезпечення для конкретних завдань та обґрунтувати доцільність впровадження нового підходу.

Для підтвердження актуальності розробки власної реалізації необхідно проаналізувати деякі популярні програмні засоби, які використовуються для формування кола на гексагональному растрі. Для цього проведемо аналіз основних функцій, переваг і недоліків програмних засобів.

Розглянемо програмне забезпечення Inkscape, що зображено на рисунку 1.1, є популярним інструментом для роботи з векторною графікою. Її основне призначення полягає у створенні та редагуванні зображень, що включають

геометричні примітиви, криві, текст і різноманітні графічні елементи [27]. Inkscape також підтримує базову роботу з растром, включаючи створення примітивів, зміну їхніх розмірів, обертання, налаштування кольорів і багатьох інших параметрів.

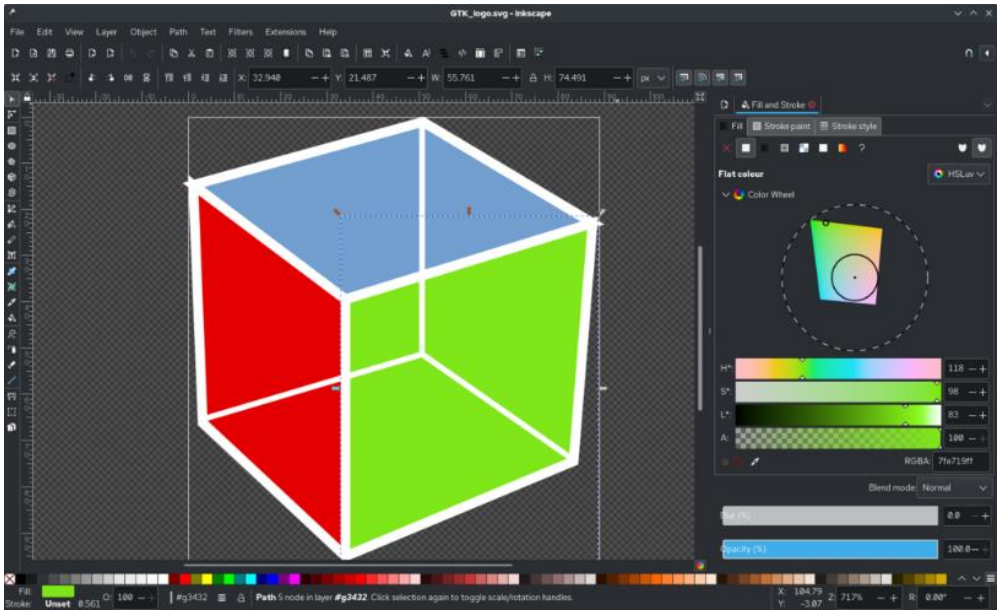

Рисунок 1.1 – Інтерфейс Inkscape

До основних переваг програми можна віднести те, що вона є безкоштовною з відкритим вихідним кодом. Крім того, Inkscape пропонує широкий набір функцій для роботи з векторною графікою, включаючи можливість експорту у різноманітні формати зображень. Простий і зрозумілий інтерфейс робить цю програму доступною навіть для користувачів без попереднього досвіду роботи з графічним програмним забезпеченням.

Водночас Inkscape має певні недоліки. Зокрема, програма не підтримує створення та роботу з гексагональним растром, що значно обмежує її застосування у спеціалізованих завданнях. Також при масштабуванні великих зображень можуть виникати помилки, що впливають на точність побудови примітивів. Це робить Inkscape менш придатним для завдань, пов’язаних із гексагональною графікою, проте її універсальність залишається значною перевагою для загальних потреб у векторній графіці.

Програма підтримує роботу з растром і забезпечує створення та обробку примітивів. В Inkscape реалізовано такі функції для роботи з графікою:

- створення нових елементів;
- редагування існуючих об’єктів;
- зміна розмірів елементів;
- обертання графічних об’єктів;
- налаштування кольорів та інших властивостей.

Основні переваги Inkscape:

- безкоштовне використання та відкритий вихідний код;
- велика кількість інструментів для обробки векторної графіки;
- можливість збереження зображень у різноманітних форматах;
- підтримка налаштування багатьох параметрів формування графічних елементів;
- інтуїтивний та зрозумілий інтерфейс для користувачів.

Недоліки Inkscape:

- відсутність підтримки формування зображень на гексагональному растрі;
- ймовірність помилок при роботі з великими масштабами.

У підсумку Inkscape є потужним і безкоштовним інструментом для роботи з векторною графікою, який пропонує широкий функціонал та зручний інтерфейс. Проте, його основним недоліком є відсутність підтримки гексагонального растру, що обмежує його використання для специфічних задач у графічних системах.

Іншим аналогом є програма Processing (рис. 1.2), що являє собою безкоштовне середовище програмування, яке орієнтоване на створення графічних зображень і візуалізацію даних. Її головною перевагою є простота у використанні, що робить програму популярною серед художників, дизайнерів і дослідників, які працюють із візуальними даними [28]. Processing підтримує роботу з растром, включаючи створення та редагування зображень за допомогою

різних бібліотек, таких як «hexagon», яка дозволяє створювати фігури на гексагональній сітці.

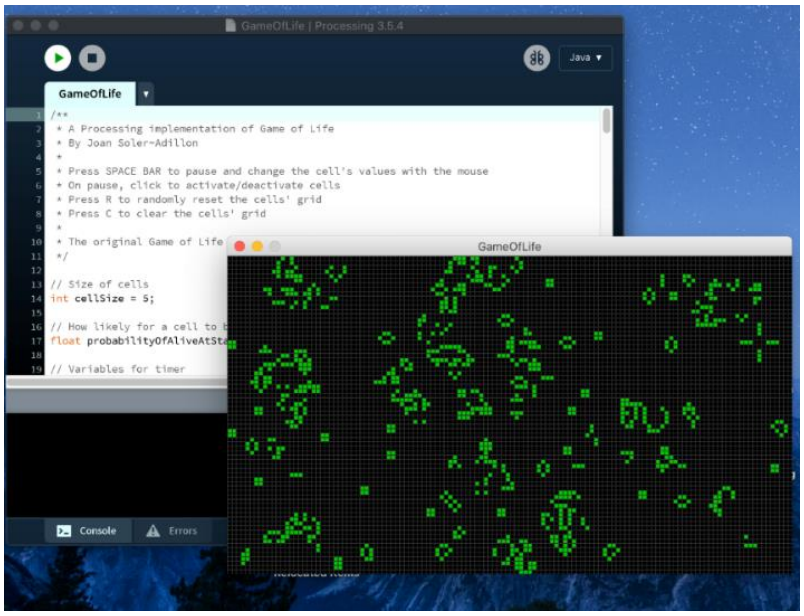

Рисунок 1.2 – Інтерфейс Processing із застосуванням формування растру

Серед ключових переваг Processing можна виділити її безкоштовність та відкритий вихідний код. Бібліотека «hexagon» забезпечує підтримку гексагонального растру, що дозволяє формувати примітиви, зокрема кола та лінії, без значних додаткових налаштувань. Простий інтерфейс і добре документовані функції роблять програму доступною для користувачів із мінімальним досвідом програмування.

Розглянемо недоліки Processing. Вона вимагає базових знань програмування для створення та редагування зображень. Крім того, його функціонал не такий потужний, як у спеціалізованих програм для роботи з векторною графікою, наприклад, Inkscape. Processing є ефективним інструментом для початкової роботи з графікою на гексагональному растрі, проте складні проєкти можуть потребувати використання більш професійного програмного забезпечення. Таким чином, програма підходить для реалізації базових функцій, але має обмеження в потужності та гнучкості для роботи з масштабними графічними завданнями.

Програмне середовище Processing надає кілька функціональних можливостей для роботи з фігурами на гексагональному растрі:

- створення графічних об’єктів із вказівкою координат центра та осей;
- зміна параметрів графіки, таких як колір і розмір;
- маніпуляції з об’єктами на растрі, включаючи їх переміщення та обертання.

Переваги Processing:

- вільний доступ і відкритий вихідний код, що дозволяє модифікувати програму під власні потреби;
- наявність бібліотеки hexagon, яка спрощує створення графічних елементів на гексагональному растрі;
- інтуїтивно зрозумілий інтерфейс, який полегшує роботу з програмою.

Недоліки Processing:

- потребує базових навичок програмування для повноцінного використання функціоналу;
- обмежена функціональність у порівнянні з потужними програмами для роботи з векторною графікою.

У підмумку можна стверджувати, що Processing є ефективним і доступним інструментом для роботи з гексагональним растром завдяки підтримці бібліотеки hexagon та простому інтерфейсу. Однак його застосування вимагає базових знань програмування, а функціональні можливості поступаються більш спеціалізованим програмам для векторної графіки.

Ще одним аналогом є спеціалізоване програмне забезпечення Hexels (рис. 1.3), яке активно використовується для створення піксельної графіки, підтримуючи гексагональний растр.

Hexels надає широкий набір можливостей, включаючи формування примітивів різного розміру, створення геометричних фігур із налаштуванням параметрів кольору та розмірів, а також інтеграцію текстових і графічних елементів [29]. Однією з ключових переваг Hexels є користувацький інтерфейс,

що полегшує роботу навіть для користувачів із мінімальним досвідом. Програма також забезпечує високу якість та деталізацію зображень.

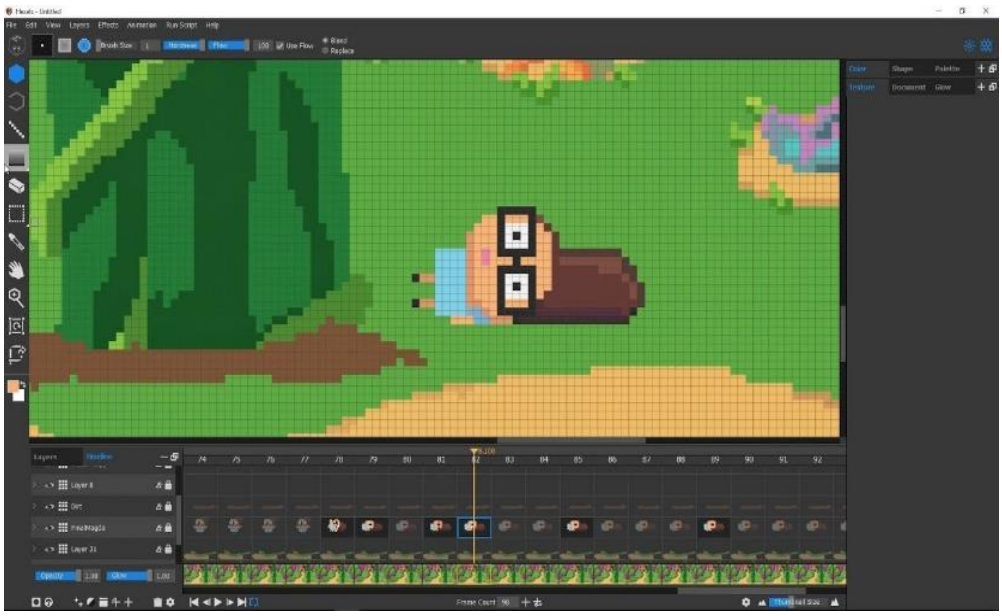

Рисунок 1.3 – Користувацький приклад формування рисунку на Hexels

Незважаючи на очевидні переваги, Hexels має низку обмежень. По-перше, вартість програми значно перевищує аналогічні рішення. Крім того, функціонал для обробки векторної графіки є обмеженим, що зменшує її універсальність. Освоєння всіх доступних інструментів може зайняти значний час, особливо для нових користувачів.

Переваги Hexels:

- інтуїтивний інтерфейс, що забезпечує зручність роботи з гексагональним растром.
- широкий набір інструментів для створення та редагування графіки;
- висока якість візуалізації створених зображень;
- підтримка налаштування параметрів кольору, розміру та інших характеристик фігур;
- можливість створення примітивів та геометричних фігур будь-якого розміру.

Недоліки Hexels:

- висока вартість програми порівняно з іншими програмами;
- обмежений функціонал для редагування та обробки векторної графіки;
- значний час, необхідний для освоєння всіх функцій та інструментів програми.

Hexels є ефективним інструментом для графічного дизайну на гексагональному растрі, однак її специфічна спрямованість та висока вартість можуть стати суттєвими факторами під час вибору програмного забезпечення для реалізації комплексних проєктів.

Таблицю порівняння наведено в таблиці 1.1.

Таблиця 1.1 – Порівняння параметрів реалізацій

Критерій	Processing	Inkscape	Hexels	HexaScreen
Налаштування власних пресетів	+	-	-	+
Швидкодія	-	+	+	+
Підтримка векторної графіки	+	+	-	+
Ціна	+	-	-	+
Складність UI/UX	+	+	+	+
Оптимальна кількість налаштувань	+	+	+	+
Кастомізація	-	+	-	+
Якість вихідного зображення	+	+	+	+
Сума	6	6	4	8

Згідно з проведеним аналізом, найкращі показники демонструє власна розробка, оскільки вона враховує специфічні потреби для роботи з гексагональним растром. Вона забезпечує можливість редагування існуючих зображень, високу якість створення графічних примітивів, оптимізацію процесу, а також широкі можливості для кастомізації під індивідуальні вимоги проєкту.

Inkscape також показує гідні результати, проте через відсутність підтримки гексагонального растру та обмеження в роботі з масштабними графічними

примітивами її функціонал залишається неповним для потреб даного дослідження. Hexels і Processing мають сильні сторони в окремих аспектах, таких як інтеграція бібліотеки hexagon (Processing) чи потужні інструменти для створення графіки (Hexels), але в цілому їх функціонал поступається можливостям власної розробки.

1.3 Аналіз методів розв’язання завдання

Для створення інформаційної технології для побудови зображень на шестикутному растрі слід дослідити математичні підходи й алгоритми, що сприяють ефективній реалізації графічних примітивів. Шестикутний растр має унікальні геометричні властивості, що вимагає адаптації традиційних методів до нової структури. Розгляд існуючих підходів дозволить виявити найкращі методики для формування фігур та їх інтеграції у загальний процес обробки.

Сучасні методи формування графічних примітивів, такі як лінії, кола та багатокутники, розроблені переважно для традиційного квадратного растру. Вони характеризуються високою точністю, але часто не враховують геометричні особливості шестикутної сітки, що призводить до втрати ефективності або якості візуалізації [30]. Саме тому необхідно адаптувати ці методи або створювати нові, які враховують переваги шестикутного растру, такі як більш рівномірне покриття площини та природніший вигляд кривих.

Крім того, шестикутний растр вимагає розробки спеціальних алгоритмів, які можуть забезпечити швидке та точне формування примітивів, мінімізуючи обчислювальні витрати та покращуючи візуальну якість зображень. У рамках цього підрозділу буде розглянуто основні підходи, що використовуються для розв’язання задачі, Їх сильні сторони й обмеження, а також встановлено перспективи для їхнього вдосконалення.

Дослідження методів і засобів формування примітивів на гексагональному растрі є важливим напрямом у розвитку сучасних графічних систем з кількох

причин [30]. Насамперед, гексагональний растр має унікальні геометричні властивості, які забезпечують більш рівномірне покриття площини, ніж традиційний квадратний растр. Це дозволяє створювати зображення з меншою кількістю видимих артефактів та покращеною якістю контурів.

Також гексагональна структура сприяє більш точному відображенню криволінійних форм, що має ключове значення для впровадження складних геометричних об'єктів. Завдяки цьому гексагональний растр знаходить застосування в таких галузях, як комп'ютерна графіка, ігрова індустрія, візуалізація даних та моделювання природних систем.

Ефективне використання цього типу растру вимагає розробки нових алгоритмів і методів формування примітивів, таких як лінії, кола та багатокутники, що враховують його специфічну геометрію. Наприклад, адаптація методів інтерполяції та антиаліайзингу дозволяє покращити якість візуалізації та уникнути дефектів зображень.

Розглянемо методи формування відрізків прямих, що можна застосувати для реалізації власної розробки.

Метод Брезенхема для шестикутного растру, незважаючи на його популярність і швидкодію, має певні недоліки, які стають очевидними при роботі з реальними задачами. Його стандартна формула була розроблена для квадратного растру, і хоча адаптація до шестикутної структури забезпечує основний функціонал, виникають труднощі з точним визначенням меж гексагонів. Це впливає на якість побудови прямих і призводить до появи артефактів, особливо на великих масштабах.

Суперсемплінг гарантує відмінну якість візуалізації, проте для шестикутного растру він також вимагає адаптації. Більшість існуючих методів суперсемплінгу орієнтовані на квадратні пікселі, і їх пряме застосування до шестикутної структури призводить до надлишкових обчислень і неефективного використання ресурсів [31]. Це створює потребу у формулюванні нових підходів, які враховують специфіку шестикутного розташування елементів.

Зважаючи на зазначені недоліки, можна зробити висновок, що адаптація існуючих формул і алгоритмів для роботи із шестикутним растром є недостатньо ефективною для багатьох практичних задач. Необхідно розробити нові підходи, які будуть враховувати геометричні особливості шестикутної решітки та забезпечувати оптимальне використання обчислювальних ресурсів. Зокрема, доцільно звернути увагу на застосування спеціалізованих формул, які дозволяють імітувати відображення гексагонів на квадратній решітці, але з точним збереженням їх геометрії та відносного розташування.

Отже, подальші дослідження мають бути зосереджені на розробці власних або адаптацію існуючих рішень алгоритмів для шестикутного растру. Це дозволить врахувати його унікальні особливості, покращити продуктивність і якість побудови графічних примітивів, а також розширити можливості використання шестикутного растру в різноманітних прикладних задачах.

Ще одним важливим аспектом при роботі із растром є формування кола або його фрагментів. На шестикутному растрі це є складним завданням, що вимагає врахування специфічних геометричних властивостей цього типу структури. Традиційні підходи, розроблені для квадратних растрових структур, часто виявляються неефективними, якщо їх без змін застосовувати у контексті шестикутного растру, тому їх адаптація є необхідною.

Одним із методів є колова інтерполяція, яка полягає у розбитті кола на сегменти з груп пікселів, розташованих у межах околу. Завдяки інтерполяції всередині цих сегментів вдається досягти плавності контурів. Однак, складність адаптації цього методу для шестикутної сітки створює додаткові виклики.

Метод апроксимації форми кола через вибір найближчих до його меж пікселів є ще одним підходом. На шестикутному растрі це може означати вибір центральної точки шестикутника або однієї з його граней. Попри швидкість реалізації, цей метод часто призводить до виникнення артефактів у вигляді «сходинок» на контурі.

Формування кола за допомогою суперпозиції накладає один на одного набори простих геометричних фігур, таких як шестикутники чи трикутники, що

дозволяє досягати високої точності, але є доволі ресурсозатратним підходом [32]. Подібний ефект забезпечує використання кривих Безьє, які дозволяють створювати гладкі та деталізовані контури, але потребують ретельних розрахунків для коректного врахування шестикутної структури.

Вибір методу залежить від конкретних завдань. Наприклад, апроксимація може бути ефективною у випадках, коли потрібна швидка обробка, тоді як метод кривих Безьє підходить для задач, де важлива максимальна якість візуалізації. Проте, жоден із методів у сучасному вигляді не забезпечує універсального рішення, оптимального для шестикутного растру.

Зазначені міркування підкреслюють необхідність розробки нових алгоритмів або адаптації існуючих рішень для формування кіл у такому специфічному середовищі [33]. Подальша робота має бути зосереджена на створенні алгоритмів, що враховують геометричні особливості шестикутників, забезпечують оптимальне використання ресурсів і досягають високої якості візуалізації. Це дозволить не тільки розширити можливості сучасних графічних систем, але й сприятиме подальшому розвитку інструментів для роботи з шестикутним растром.

Також необхідно зазначити, що питання оптимізації є специфічним для гексагонального растру, адже вимагає особливого підходу до візуалізації графічних примітивів при перетворенні зображення із квадратної решітки у шестикутну. Основною метою є підвищення якості відображення примітивів та забезпечення продуктивності обчислень. Це завдання має вирішити візуальні аспекти з такими недоліками, як нерівні краї об'єктів та значні обчислювальні витрати.

Антиаліайзинг є ключовим інструментом для усунення візуальних дефектів на межах примітивів. Він дозволяє створювати плавні переходи між кольорами пікселів на межах об’єктів та фону. Для шестикутного растру цей процес вимагає адаптації через його незвичайну геометрію. Метод суперсемплінгу, зокрема, дозволяє зменшити ефект «сходинок» на краях за рахунок додаткових обчислень кольору субпікселів [34]. Альтернативою є

субпіксельна інтерполяція, яка, хоч і менш ресурсомістка, також сприяє покращенню якості зображень.

Продуктивність є ще одним важливим аспектом. Для підвищення швидкодії доцільно застосовувати мемоізацію, що дозволяє уникати повторних розрахунків, та паралельну обробку, яка значно прискорює процес формування зображень. Адаптивна деталізація також є ефективним рішенням: об'єкти, що знаходяться далі, можуть відображатися з меншою кількістю деталей, не впливаючи на загальну якість сцени.

Оптимізація програмного коду, зокрема використання апаратного прискорення, сприяє зниженню обчислювальних витрат. Крім того, алгоритми, спеціально адаптовані для шестикутного растру, забезпечують точніші й швидші обчислення. Такі рішення, як прискорене визначення сусідніх гексагонів або спрощені методи побудови примітивів, демонструють значний потенціал у цій сфері [33].

1.4 Постановка задач розробки інформаційної технології для оптимізації побудови зображень на шестикутному растрі

Шестикутний растр, завдяки своїм геометричним особливостям, має значний потенціал у сфері графічного моделювання, візуалізації даних та комп’ютерної графіки [30]. Однак існуючі інформаційні технології та програмні інструменти часто не враховують специфіку цього растру, обмежуючи його використання в прикладних і наукових завданнях.

Розробка нової інформаційної технології для роботи з шестикутним растром є актуальною через декілька ключових аспектів. По-перше, гексагональна структура забезпечує кращу апроксимацію кривих і гладкість візуалізації порівняно з квадратним растром. По-друге, шестикутна сітка дозволяє ефективніше покривати площину, зменшуючи кількість зайвих обчислень, пов'язаних із розташуванням елементів. По-третє, адаптація

існуючих алгоритмів для роботи з шестикутним растром створює низку технічних викликів, зокрема щодо визначення координат, оптимізації обчислень та збереження високої якості візуалізації.

Основні виклики, пов’язані з роботою на шестикутному растрі, включають необхідність адаптації математичних моделей для представлення графічних примітивів, розробку алгоритмів для точного формування фігур, а також забезпечення продуктивності при великому обсязі обчислень. Існуючі підходи, розроблені для квадратного растру, не можуть бути безпосередньо застосовані до шестикутного растру через його геометричні відмінності. Це потребує суттєвого переосмислення та вдосконалення методів обробки графіки.

Для вирішення цих проблем необхідно створити нову інформаційну технологію, яка забезпечить високу продуктивність і точність, а також адаптивність до різних графічних задач. Така технологія має базуватися на сучасних математичних моделях, ефективних алгоритмах та оптимізованих програмних рішеннях, які відповідають потребам сучасної графічної індустрії.

Результатом має стати платформа, здатна виконувати широкий спектр задач, пов’язаних із формуванням та обробкою зображень на шестикутному растрі, забезпечуючи при цьому високу якість візуалізації, зручність використання і можливість інтеграції з іншими графічними системами.

Тому спочатку необхідно створити концептуальну модель інформаційної технології, яка включатиме опис функціональних можливостей, логіки роботи з графічними примітивами, взаємодію з користувачем та обробку зображень на шестикутному растрі. Ця модель повинна враховувати специфіку геометричної структури шестикутного растру та забезпечувати адаптивність для різних задач.

Наступний етап полягає у розробці ефективних алгоритмів формування базових графічних примітивів, таких як лінії, багатокутники та інші фігури, з урахуванням особливостей шестикутної структури. Важливим аспектом буде забезпечення продуктивності алгоритмів і точності відображення геометричних форм.

Важливо також приділити увагу для створення оптимізованих алгоритмів формування примітивів на шестикутному растрі. Через відмінності між шестикутною та квадратною решітками традиційні алгоритми формування не можуть бути безпосередньо використані, що потребує розробки спеціалізованих рішень для забезпечення візуальної гладкості та точності контурів.

Наступним кроком є реалізація програмного забезпечення для підтримки роботи з шестикутним растром. Це включає створення відповідних модулів, класів і функцій, які дозволять зручно формувати, редагувати та візуалізувати графічні примітиви. Інтерфейс користувача має бути інтуїтивно зрозумілим і функціонально насиченим.

Окрім цього, тестування програмного продукту є обов'язковим етапом розробки. Воно дозволить оцінити функціональність, продуктивність та надійність створених алгоритмів і програмних рішень. Результати тестування мають стати основою для вдосконалення системи та підвищення її якості.

Отже, виконання визначених завдань сприятиме забезпеченню створення інформаційної технології, яка дозволить оптимізувати побудову зображень на шестикутному растрі та покращить якість графічного моделювання.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки до розділу

У першому розділі було розглянуто процес формування зображень та графічних примітивів на шестикутному растрі, виконано детальний порівняльний аналіз існуючих рішень, а також визначено сучасні методи побудови примітивів. Окрему увагу було приділено постановці завдань для створення нової інформаційної технології, яка враховує специфіку цього типу растру.

Результати дослідження продемонстрували переваги шестикутного растру над традиційним квадратним. Завдяки рівномірнішому покриттю площини,

меншій кількості артефактів і природнішому відображенню криволінійних форм, шестикутний растр є перспективною структурою для графічних систем. Однак, через його специфічну геометрію виникає потреба в адаптації існуючих методів і алгоритмів. Існуючі програмні засоби, такі як Inkscape, Processing і Hexels, хоча й забезпечують базові можливості, не відповідають повністю вимогам ефективної роботи з шестикутним растром. Це підтверджує необхідність розробки власної інформаційної технології, орієнтованої на конкретні потреби даної області.

Аналіз методів формування графічних примітивів виявив обмеження традиційних підходів. Методи, такі як Брезенхема та суперсемплінг, вимагають адаптації через труднощі в точному визначенні меж гексагонів і надмірні обчислювальні витрати. Подібна ситуація спостерігається і з методами побудови кіл, які у сучасному вигляді не забезпечують оптимальної точності та якості візуалізації для шестикутного растру. Таким чином, створення спеціалізованих алгоритмів є необхідністю для досягнення високої якості та продуктивності.

Було окреслено основні напрями розробки, які включають створення моделі роботи із шестикутним растром, проектування нових алгоритмів для формування графічних примітивів, розробку зручного інтерфейсу для роботи з користувачами, а також тестування та оптимізацію продуктивності програмного забезпечення. Запропоновані підходи сприятимуть покращенню якості візуалізації, зменшенню обчислювальних витрат і забезпеченню гнучкості при роботі із графічними об’єктами.

Таким чином, реалізація поставлених завдань дозволить створити ефективну інформаційну технологію для роботи із шестикутним растром, яка буде відповідати сучасним потребам в області графічного моделювання, комп’ютерної графіки та візуалізації даних. Це відкриває нові можливості для використання шестикутного растру в прикладних і наукових задачах, розширюючи перспективи його застосування в різноманітних галузях.

2 РОЗРОБКА АЛГОРИТМІВ І МАТЕМАТИЧНИХ МОДЕЛЕЙ ДЛЯ РОБОТИ З ШЕСТИКУТНИМ РАСТРОМ

2.1 Удосконалення методу оцінювальної функції для побудови прямих на шестикутному растрі

Розробка методів формування прямих на шестикутному растрі є важливим етапом для оптимізації графічних систем, особливо у застосуваннях, де потрібна висока точність і природність візуалізації. Геометрична структура гексагонального растру створює додаткові виклики у порівнянні з квадратною сіткою, оскільки традиційні методи формування ліній не можуть бути використані без адаптації [2]. Цей розділ зосереджений на адаптації існуючих методів формування ліній до особливостей шестикутної сітки, а також на демонстрації результатів модифікації через візуальні приклади. Важливо показати, як запропоновані підходи усувають недоліки існуючих рішень і забезпечують підвищення точності, швидкодії та якості графічної візуалізації.

Гексагональний растр базується на правильних шестикутниках, які тісно заповнюють площину без утворення порожніх зон. Запропонована геометрія дозволяє мінімізувати повторювані артефакти, що часто виникають на традиційному квадратному растрі, особливо при відображенні діагональних ліній чи криволінійних об'єктів. Шестикутна сітка забезпечує більш однорідне покриття площини, зменшуючи кількість сусідніх точок, які необхідно враховувати під час побудови графічних примітивів.

Порівняно із квадратною сіткою, шестикутний растр має кілька переваг. Наприклад, кожен піксель у гексагональному растрі має шість сусідів замість чотирьох, що забезпечує більш природну апроксимацію кривих і покращує якість візуалізації. Крім того, гексагональний растр усуває проблему "сходинкового ефекту" (aliasing), яка характерна для квадратних пікселів, особливо на діагональних лініях. Завдяки цьому лінії на шестикутній сітці виглядають більш плавно і реалістично. Водночас, реалізація алгоритмів на

гексагональній сітці є складнішою, оскільки відсутні прямі горизонтальні чи вертикальні осі, як у квадратній системі.

Використання координатних приростів є ключовим елементом у моделюванні траєкторій на шестикутному растрі [26]. Через особливості розташування пікселів у вигляді шестикутників обчислення приростів координат вимагає адаптації традиційних формул. У квадратній сітці прирости визначаються лінійними співвідношеннями між сусідніми пікселями, тоді як у гексагональній структурі необхідно враховувати діагональні зв'язки. Це додає складності, оскільки діагональні прирости не є симетричними відносно обох осей.

Математичне моделювання гексагональної сітки використовує координати центральної точки гексагона як базу для обчислень. Враховуючи, що кожен сусідній піксель знаходиться на відстані, пропорційній висоті шестикутника, для коректного моделювання траєкторій необхідно визначити точну залежність між зміщеннями по кожній з осей. Ці залежності забезпечують правильне відображення ліній і примітивів, зберігаючи точність і природність їхнього вигляду на гексагональному растрі.

Артефакти на межах гексагонів є однією з ключових проблем при роботі з шестикутним растром, оскільки вони можуть суттєво знижувати якість візуалізації. Через унікальну геометрію гексагональної структури, межі між окремими елементами растру можуть проявлятися у вигляді нерівностей або розривів, особливо при відображенні діагональних ліній чи криволінійних форм. Ці артефакти виникають через недостатню адаптацію алгоритмів, спроектованих для квадратних сіток, що унеможливлює точне врахування особливостей шестикутного растру.

Недостатня точність обчислень у методах, адаптованих із квадратного растру, також є суттєвим викликом. Традиційні алгоритми, такі як метод оцінювальної функції чи цифровий диференційний аналізатор, розроблялися з урахуванням простоти квадратної геометрії, де крокові зміщення легко обчислюються відносно координатних осей. При їх перенесенні на шестикутний

растр виникають помилки через необхідність врахування додаткових векторів зміщення, які не є симетричними або лінійними, як у квадратній сітці. Це призводить до похибок у розрахунках координат, що впливає на точність побудови графічних примітивів.

Високі обчислювальні витрати є ще однією проблемою, яка значно впливає на ефективність роботи з шестикутним растром. У порівнянні з квадратним растром, шестикутна структура вимагає більш складних алгоритмів для визначення сусідніх елементів та обчислення точок траєкторії. Необхідність урахування діагональних зв'язків, асиметричних приростів і складних геометричних співвідношень збільшує обсяг обчислень і ресурсів, які витрачаються на побудову графічних елементів. Це може стати критичним фактором при роботі із великими обсягами даних або в реальному часі, де продуктивність має ключове значення.

Таким чином, для покращення роботи з шестикутним растром необхідно розробляти алгоритми, які враховують специфічні особливості цієї геометрії. Це дозволить мінімізувати артефакти, підвищити точність обчислень і знизити обчислювальні витрати, забезпечуючи при цьому високу якість та ефективність візуалізації.

Оцінювальна функція є базовим методом для формування ліній завдяки своїй універсальності, простоті реалізації та ефективності. Її основна перевага полягає в можливості визначати напрямок кроку в траєкторії за допомогою аналізу знаків функції, що дозволяє оптимально керувати процесом побудови ліній. Для шестикутної сітки, де структура пікселів відрізняється від традиційного квадратного растру, цей метод має бути адаптований із врахуванням специфіки геометрії. Це обумовлює вибір оцінювальної функції як основи для розробки алгоритмів, які ефективно працюють із гексагональним растром.

Математичні перетворення є необхідним інструментом для врахування унікальних властивостей шестикутної сітки. На відміну від квадратного растру, де зміщення між пікселями є лінійним і ортогональним, шестикутна структура

вимагає врахування діагональних з'єднань і складних співвідношень між координатами. Для коректного визначення траєкторій необхідно використовувати математичні моделі, які описують зв'язки між центрами гексагонів і передбачають точні розрахунки координат для кожного кроку. Зокрема, такі моделі повинні враховувати співвідношення між осьовими приростами та діагональними відстанями, що забезпечить точність побудови ліній.

Для створення ліній на графічних растрах використовуються кілька основних підходів, серед яких найбільш популярними є метод прямого обчислення, цифровий диференційний аналізатор (ЦДА) та метод оцінювальної функції.

Спосіб прямого розрахунку базується на використанні рівняння прямої виду $Y = Y_n + \Delta Y * \frac{X}{\Delta X}$. У випадках, коли $\Delta X \geq \Delta Y$, у рівняння підставляється поточна координата X, після чого обчислюється відповідне значення Y. Якщо ж $\Delta X < \Delta Y$, підставляється Y, і обчислюється X. Головним недоліком цього методу є необхідність виконання обчислень із діленням, що суттєво впливає на продуктивність. Це обмежує його застосування в задачах, де висока швидкодія є критично важливою.

Метод цифрового диференційного аналізатора передбачає заміну обчислень множення та ділення на простіші операції додавання. На етапі підготовки визначається відношення $\frac{\Delta Y}{\Delta X}$, яке згодом використовується для ітераційного накопичення координат. Такий підхід підвищує продуктивність обчислень порівняно з методом прямого розрахунку, але його точність залишається обмеженою в задачах із великим числом проміжних кроків.

Спосіб оцінювальної функції є універсальним і широко застосовуваним для формування ліній. Його основною перевагою є можливість швидкого визначення напрямку кроку, аналізуючи знак спеціально розрахованої функції. Наприклад, якщо точка знаходиться вище лінії, значення функції буде додатним, і наступний крок виконується по осі X. Якщо точка розташована нижче, значення

буде від’ємним, і крок виконується по осі Y. Доцільно використовувати для діагональних приростів, коли формується комбінований крок по обох осях.

Для адаптації способу ОФ до гексагонального растру слід врахувати специфічну геометрію шестикутників. Загальна формула оцінювальної функції для прямої, заданої приростами ΔX і ΔY, може бути виражена у вигляді:

$$OF_i = \frac{y_i}{x_i} - \frac{\Delta y}{\Delta x} = \frac{y_i * \Delta x - x_i * \Delta y}{x_i * \Delta x} \quad (2.1)$$

Оскільки знаменник у цій формулі не впливає на знак функції, для спрощення обчислень можна використовувати модифіковану версію:

$$OF_i = y_i * \Delta x - x_i * \Delta y \quad (2.2)$$

Для визначення наступного кроку потрібно лише оцінити знак OF_i. Якщо $OF_i \geq 0$, крок виконується по осі OX. Коли $OF_i < 0$ – крок слід виконати по обох осях одночасно, що відповідає діагональному переміщенню. Приклад формування відрізку прямої відповідно зазначених характеристик зазначено на рисунку 2.1.

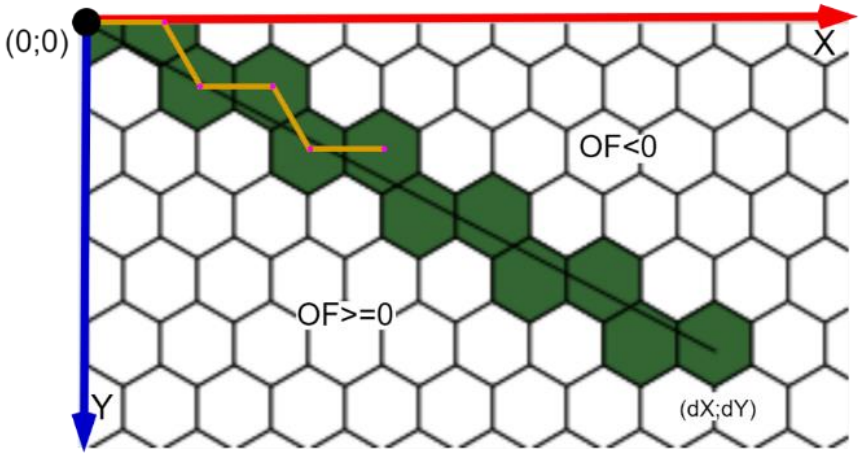

Рисунок 2.1 – Ітеративний процес побудови лінії

Якщо точка наступної ітерації на траєкторії розташована вище лінії b, то результат обрахунку OF_i є невід'ємним ($OF_i \geq 0$), і наступний крок необхідно виконувати вздовж осі OX. У випадку, коли точка знаходиться нижче лінії b ($OF_i < 0$), наступний крок виконується одночасно по обох осях, що відповідає діагональному переміщенню.

Для оптимізації розрахунків слід зазначити, що знаменник дробу у формулі оцінювальної функції:

$$OF_i = \frac{y_i}{x_i} - \frac{\Delta y}{\Delta x} = \frac{y_i * \Delta x - x_i * \Delta y}{x_i * \Delta x} \quad (2.3)$$

не впливає на визначення знака функції. Тому знаменник $x_i * \Delta x$ можна ігнорувати, і формулу оцінювальної функції можна спростити до вигляду:

$$OF_i = y_i * \Delta x - x_i * \Delta y \quad (2.4)$$

Це суттєво знижує обчислювальну складність, оскільки виключає необхідність виконання операцій ділення. Така модифікація надає можливіть пришвидшити виконання алгоритму без втрати точності, що є критично важливим при роботі з великими об'ємами графічних даних або складними траєкторіями.

Щоб визначити цільове значення OF_iпри кроці вздовж осі OX, потрібно врахувати, що координати по OY залишаються незмінними, а координата по вісі абсцис збільшується на одиницю: $x_{i+1} = x_i + 1$.

Формула для обчислення нового значення OF_i наступної ітерації формування кроку слід записати як:

$$OF_{i+1} = y_i * \Delta x - (x_i + 1) * \Delta y = y_i * \Delta x - x_i * \Delta y - \Delta y = OF_i - \Delta y \quad (2.5)$$

Розглянемо фрагмент шестикутного піксела, як це зображено на рисунку 2.2, де можна визначити співвідношення між координатами x і y на основі геометричних властивостей гексагонального растру.

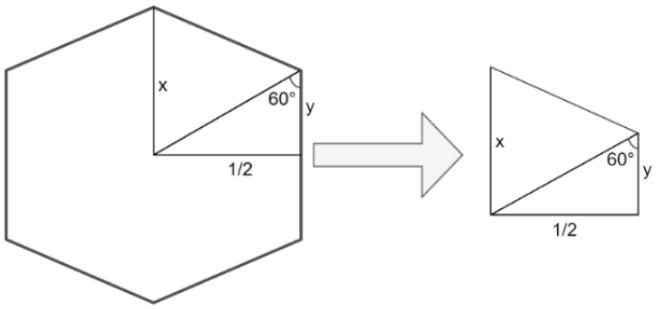

Рисунок 2.2 – Підготовка гексагону для імітації на звичайному екрані з квадратною решіткою пікселів

Для цього використовуються тригонометричні залежності, що відповідає співвідношенню між стороною трикутника та його гіпотенузою:

$$\sin 60° = \frac{\sqrt{3}}{2} = \frac{1}{2} * x \tag{2.6}$$

Якщо x є гіпотенузою, а y – протилежним катетом, то $\sin 60° = \frac{y}{x}$. Звідси $y = x * \frac{\sqrt{3}}{2}$. Спрощуючи, отримаємо $x = 2 * y$. Співвідношення між x і y дозволяє використовувати цю залежність для розрахунків у шестикутній геометрії при визначенні координат точок у квадратному растрі.

Отже, необхідно визначити нове значення оцінювальної функції при здійсненні кроку вздовж осі OX. У процесі формування крокової траєкторії координати по осі OY залишаються незмінними, тоді як координати по осі OX збільшуються на 1, тобто $x_{i+1} = x_i + 1$.

Для підвищення швидкості обчислень можна застосувати формулу:

$$OF_{i+1} = y_i * \Delta x - (x_i + 1) * \Delta y = y_i * \Delta x - x_i * \Delta y - \Delta y = OF_i - \Delta y \tag{2.7}$$

Аналогічно розраховується нове значення оцінювальної функції у разі одночасного кроку по осях OX та OY. У цьому випадку координата вздовж осі OX зростає на$\frac{1}{2}$, а по осі OY збільшується на $\frac{3}{2\sqrt{3}}$.

Відповідно до рисунку 2.2, можна встановити співвідношення між x та y: $\sin 60° = \frac{1}{2} * x;\ \ \frac{\sqrt{3}}{2} = \frac{1}{2} * x\ ;\ x = \frac{1}{\sqrt{3}}\ ;\ x = 2 * y$. Таким чином, отримуємо співвідношення $x = 2 * y$.

Оскільки $y = \frac{1}{2\sqrt{3}}$, то можна стверджувати що нове значення OY можна визначити як $OY = 3y = \frac{1}{2\sqrt{3}}$. Крок визначення y приведений на рисунку 2.3.

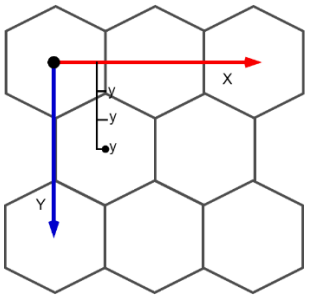

Рисунок 2.3 – Позиція центру гексагона

Координати крокової траєкторії вздовж осі OY визначаються з урахуванням того, що на гексагональному растрі відстань між центрами сусідніх пікселів становить одиницю. Таким чином, під час здійснення діагонального кроку координата по осі OX збільшується на $\frac{1}{2}$, а координата по осі OY — на $\frac{3}{2\sqrt{3}}$. У зв’язку з цим нове значення оцінювальної функції розраховується за формулою:

$$OF_{i+1} = (y_i + \frac{3}{2\sqrt{3}}) * \Delta x - (x_i + \frac{1}{2}) * \Delta y = y_i * \Delta x - x_i * \Delta y - \frac{\Delta y}{2} = OF_i + \frac{3*\Delta x}{2\sqrt{3}} - \frac{\Delta y}{2} \quad (2.8)$$

Запропоновані формули та методи обчислення є ефективними для інтерполяції відрізків прямих, які мають кути нахилу в межах від 0° до 60° щодо осі OX. Використання цих формул забезпечує точне визначення координат точок

траєкторії на гексагональному растрі, а також формування відрізків прямих із високою точністю та швидкістю.

На рисунку 2.5 зображено формування відрізка з кутами нахилу в діапазоні від 60° до 90°. Таким чином одинарні кроки по осі OX не застосовуються. Тому траєкторія формується за допомогою одночасних кроків уздовж обох осей, що відповідає діагональному руху. Це дозволяє точно визначати положення точок на шестикутному растрі, а також забезпечує ефективне формування лінії віжплвідно заданому куту нахилу.

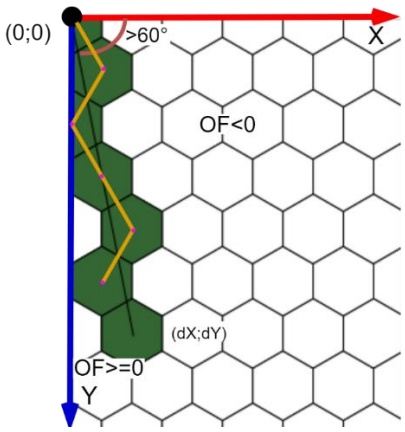

Рисунок 2.4 – Побудова прямої від 60° до 90°

Формування відрізків прямих із кутами нахилу в межах від 60° до 90° потребує застосування адаптованих правил покрокового переміщення. У випадках, коли значення оцінювальної функції $OF_i \geq 0$, здійснюється діагональний крок. При цьому координата по осі OX змінюється відповідно до формули $x_{i+1} = x_i + \frac{1}{2}$, а координата по осі OY визначається як: $y_{i+1} = y_i + \frac{3}{2\sqrt{3}}$.

За умови $OF_i < 0$ при формуванні діагонального кроку точка по OX буде обраховуватися як: $x_{i+1} = x_i - \frac{1}{2}$, а по OY: $y_{i+1} = y_i + \frac{3}{2\sqrt{3}}$. Оцінювальна функція при $OF_i < 0$:

$$OF_{i+1} = (y_i + \frac{3}{2\sqrt{3}}) * \Delta x - (x_i + \frac{1}{2}) * \Delta y = y_i * \Delta x + \frac{3\Delta x}{2\sqrt{3}} - x_i * \Delta y - \frac{\Delta y}{2} = OF_i + \frac{3*\Delta x}{2\sqrt{3}} - \frac{\Delta y}{2} \quad (2.9)$$

При формуванні відрізків прямих з кутами нахилу від 60° до 90°, покрокове формування відрізка буде відбуватися за новими правилами. Коли оцінювальна функція $OF_i >= 0$, виконується діагональний крок, і координата по осі OX змінюється за формулою $x_{i+1} = x_i + \frac{1}{2}$, а по осі OY – $y_{i+1} = y_i + \frac{3}{2\sqrt{3}}$.

Для реалізації методу формування прямої на шестикутному растрі у програмному середовищі створено відповідний алгоритм, як це зображено на рисунку 2.5.

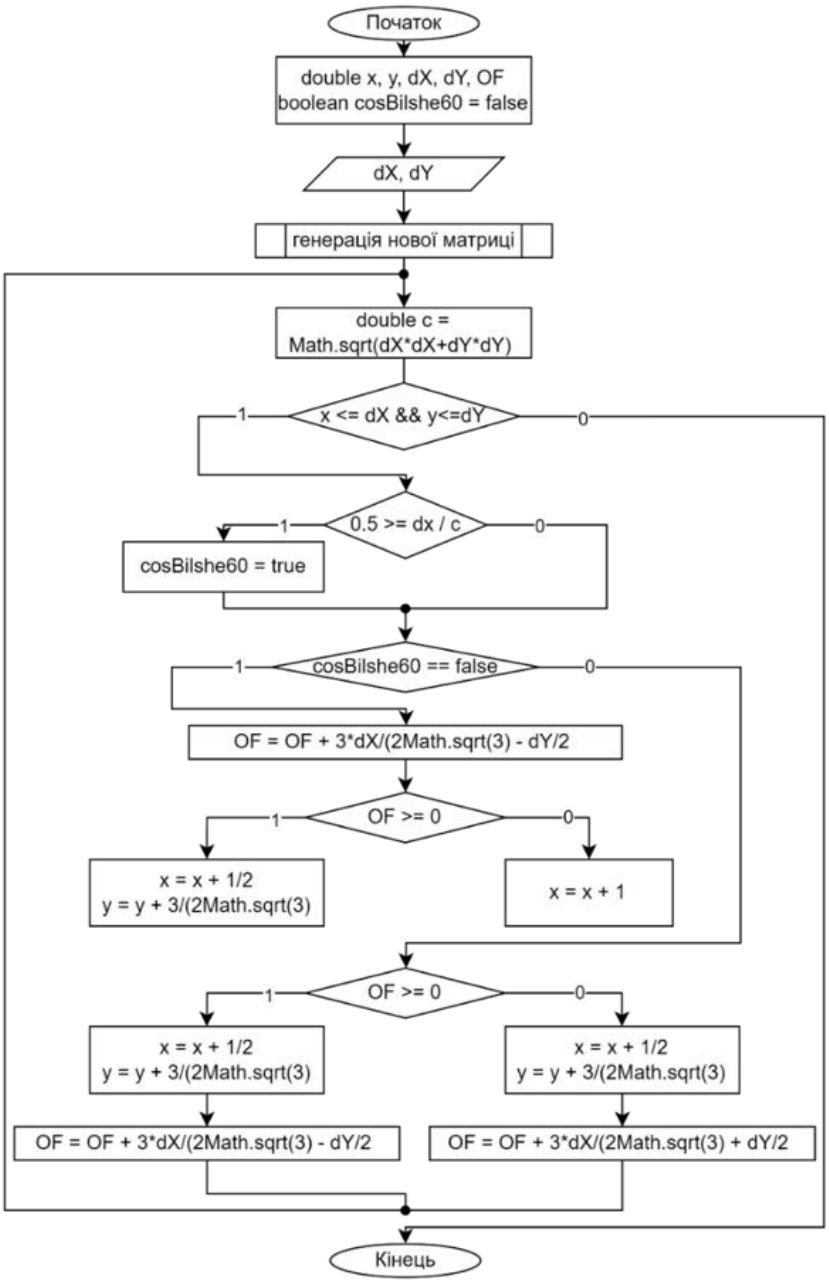

Рисунок 2.5 – Алгоритм побудови відрізка на гексагональному растрі

Інформація про програмну розробку та функціонування описані у третьому розділі роботи, а програмний код наведено в додатку В. Отриманий результат формування лінії на гексагональному екрані відповідно до алгоритму представлений на рисунку 2.6.

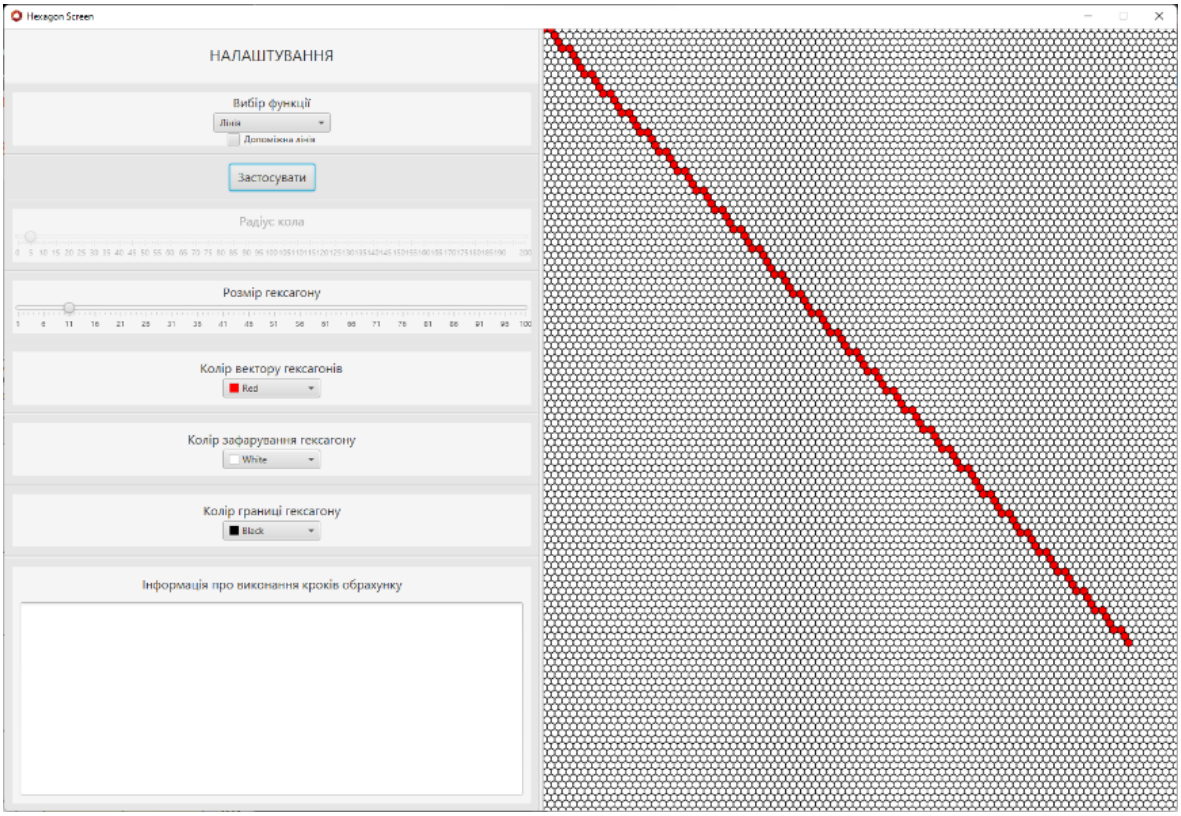

Рисунок 2.6 – Формування відрізка зі розміром гексагону в 11 пікселів

Реалізований програмний модуль дозволяє користувачеві створювати гексагональні комірки різних розмірів. Для верифікації правильності роботи алгоритму запропоновано використовувати координати, які легко перевірити вручну. Збільшення розміру гексагонів на екрані забезпечує зручність спостереження та перевірки алгоритму візуально.

У програмі встановлені значення за замовчуванням для більшості параметрів, що дозволяє користувачам відразу оцінити роботу алгоритму без додаткових налаштувань. Проте, якщо виникне потреба, користувач може змінювати положення кінцевого пікселя, вибираючи нову точку на екрані за допомогою натискання кнопки миші на відповідний гексагон робочого поля.

Після вибору нової кінцевої точки алгоритм автоматично будує новий відрізок, визначаючи координати кінцевого пікселя. У результаті на екрані відображається імітація гексагональної матриці з побудованим відрізком, як це зображено на рисунку 2.6. Такий підхід забезпечує інтерактивність і гнучкість роботи програми, що сприяє її ефективному використанню.

Таким чином, було розроблено модифікований метод і алгоритм формування прямих на гексагональному растрі. Створений метод інтегровано у програмний модуль із візуалізацією "HexaScreen", який надає можливість формувати відрізки на основі гексагональної матриці, використовуючи стандартну декартову систему координат. Цей підхід дозволяє досягти високої точності й ефективності у формуванні графічних лінійних елементів, що сприяє розширенню можливостей використання системи в різноманітних задачах.

2.2 Розробка методу колової інтерполяції на гексагональному растрі

У сфері комп'ютерної графіки завдання інтерполяції дуг кола є важливим, оскільки коло виступає одним із базових графічних примітивів поряд із прямими відрізками та еліпсами. Його значення зумовлене частим використанням у створенні графічних зображень, які здебільшого формуються на основі геометричних примітивів.

Коло широко застосовується у різноманітних графічних системах і програмах, однак його відтворення на дискретному квадратному растрі часто супроводжується низькою якістю через обмеження роздільної здатності та специфіку методів інтерполяції. Для усунення цих недоліків розглядаються альтернативні типи растрів, серед яких гексагональний растр, також відомий як стільниковий. Ця структура має переваги у забезпеченні точності й ефективності, сприяючи високій якості відтворення кіл та інших геометричних форм.

Прямий підхід до формування кіл, заснований на використанні рівняння кола, не став популярним через його складність і високу обчислювальну вартість.

Замість цього перевага надається методам, що базуються на вирішенні диференціальних рівнянь та застосуванні цифрових інтеграторів. Такі методи передбачають роботу двох цифрових інтеграторів із перехресними від'ємними зворотними зв'язками, що дозволяє досягти ефективнішого відтворення кіл на різних типах растрів, включно з гексагональним.

Реалізації методів, що базуються на цифрових інтеграторах, та їх застосування обмежується значною похибкою. Це також негативно впливає на точність результату. Важливим недоліком такого підходу є забезпечення постійної швидкості руху точки по траєкторії незалежно від радіуса кола, що може суттєво вплинути на продуктивність інтерполятора, особливо для великих радіусів.

З огляду на зазначені обмеження, були розроблені інші підходи, які поєднують високу точність і ефективність обчислень. Основна мета таких методів полягає у забезпеченні точного відтворення кола з мінімальними похибками та оптимальною продуктивністю. Серед популярних підходів для формування кіл використовується кусково-лінійна апроксимація, що дозволяє розділити коло на сегменти та апроксимувати кожен із них відрізком прямої. Проте цей метод вимагає використання значної кількості сегментів для досягнення необхідної точності, що, своєю чергою, збільшує обчислювальні витрати та додає похибки у вигляді неточності апроксимації дуг.

Методи, які для обрахунку використовують значення OF_i, вирішують зазначені раніше проблеми та забезпечують точне формування траєкторії. У випадку колової інтерполяції оцінювальна функція для точки з координатами (x,y) визначається як $OF_i = ({x_i}^2 + {y_i}^2) - R^2$, де R — радіус кола. Ця функція дає змогу легко визначити положення точки відносно кола: від'ємне значення OF_i свідчить, що точка знаходиться всередині кола, додатне — зовні, а нульове — що точка лежить на колі. Застосування таких методів забезпечує високу точність відтворення та мінімізує похибки навіть при невеликих радіусах.

Такий підхід дозволяє уникнути недоліків, пов'язаних з кусково-лінійною апроксимацією, і забезпечує більш точне формування кіл без додаткової похибки апроксимації дуги.

Для спрощення обчислень інтерполяція обмежується однією чвертю декартової системи координат, що дозволяє використати властивості симетрії, як це зображено на рисунку 2.7. Усі точки кола можна відобразити симетрично відносно осей координат і початку системи відліку, що значно скорочує обсяг необхідних обчислень. Такий підхід дозволяє ефективно отримувати координати точок кола, не обчислюючи їх для кожного сектора окремо.

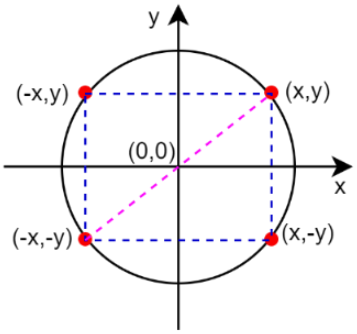

Рисунок 2.7 – Розміщення точок на колі

Для реалізації побудови кола на гексагональному растрі використовується алгоритм, що представлений на рисунку 2.8. Головною особливістю цього алгоритму є врахування специфіки гексагональної структури растру, де прирости координат залежать від поточного розташування точки у координатному просторі. Це забезпечує точне відтворення дуги кола з мінімальними похибками.

Алгоритм охоплює два основні сценарії обчислень, які залежать від кута нахилу дуги кола:

– інтерполяція дуги у секторі від 0° до 30°, де прирости координат по осях визначаються із врахуванням меншого нахилу траєкторії;

– інтерполяція дуги у секторі від 30° до 90∘, де прирости координат змінюються відповідно до збільшення кута нахилу.

Радіус кола визначається як відстань між центром кола (початковою точкою) та правим краєм кола на горизонтальній осі.

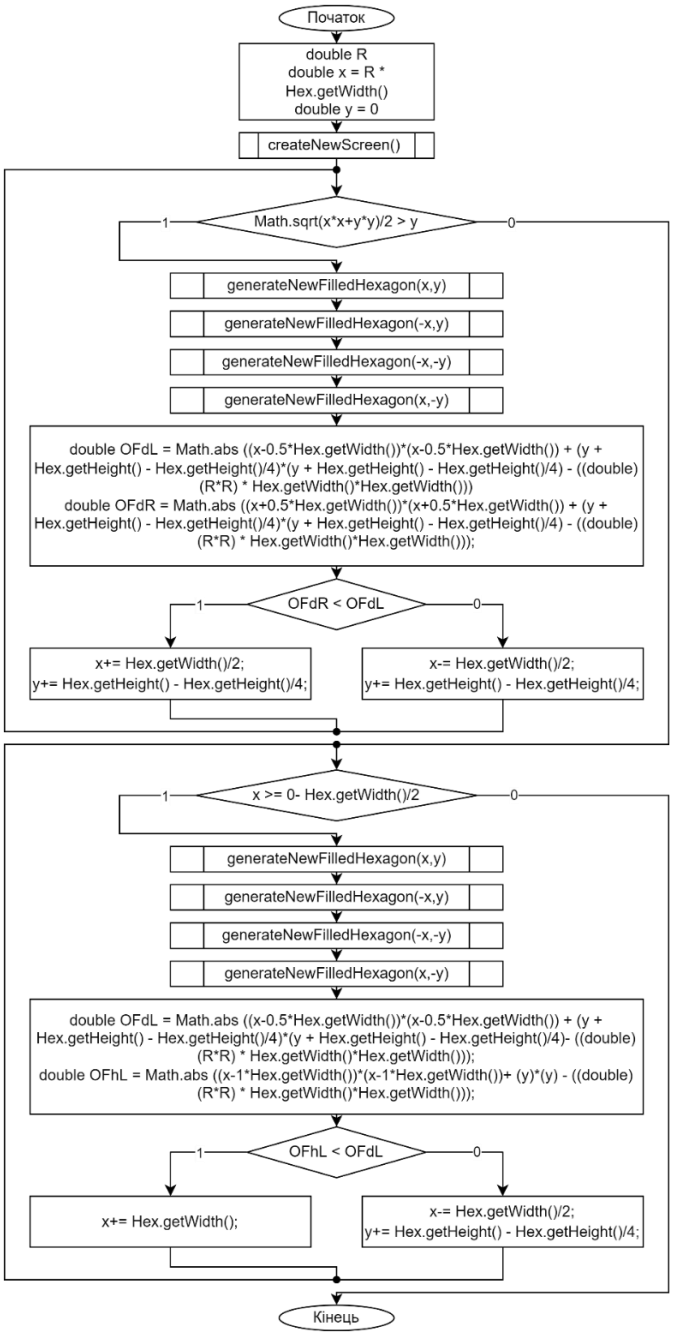

Рисунок 2.8 – Алгоритм формування кола

Використовуючи отримані значення координат X та Y, виконується виклик функції createHexagon(x, y) для формування гексагонального пікселя у відповідній точці. Детальний опис реалізації функції та алгоритму представлено у розділі 3. Після цього метод повторно викликається з іншими комбінаціями

координат: (−x,y)(-x, y)(−x,y), (x,−y)(x, -y)(x,−y) та (−x,−y)(-x, -y)(−x,−y), як це продемонстровано на рисунку 2.8.

Для визначення нового положення точки перевіряється, чи знаходиться попередня точка на дузі, що відповідає куту 30° відносно вісі абсцис і центру кола. Замість використання тригонометричних функцій, таких як синус чи косинус, для визначення кута, проводиться аналіз висоти точки та її відстані до центру кола. Це дозволяє суттєво спростити обчислення, як показано на рисунку 2.9.

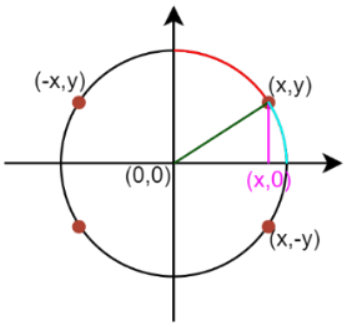

Рисунок 2.9 – Приклад для обрахунків сектору до 30°

Розроблений алгоритм виконується без використання зовнішніх бібліотек, забезпечуючи більшу незалежність та гнучкість у реалізації. Нове значення оцінювальної функції обчислюється шляхом порівняння висоти та відстані для визначення наступної точки. Цей підхід дозволяє сформувати одну чверть кола з мінімальною похибкою.

Порівняльний аналіз колової інтерполяції на гексагональному та квадратному растрах свідчить про значно вищу якість відтворення на гексагональному растрі. Відповідні результати ілюструються на рисунку 2.10, що демонструє переваги розробленого методу у точності та візуальній якості відтворення кола.

Формування частини кола в діапазоні від 0° до 30° передбачає два можливі напрями кроків по пікселях: діагонально праворуч або діагонально ліворуч.

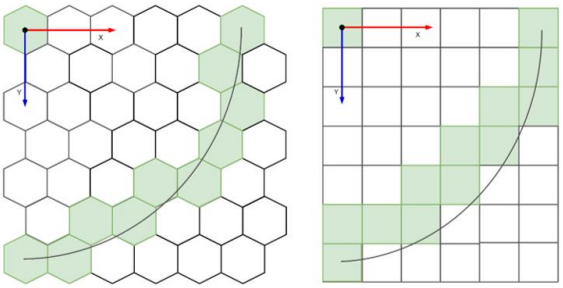

Рисунок 2.10 – Порівняння кола
ліворуч – на гексагональному растрі, праворуч – на квадратному

Схематичне зображення можливих напрямів кроків у цьому секторі наведено на рисунку 2.11.

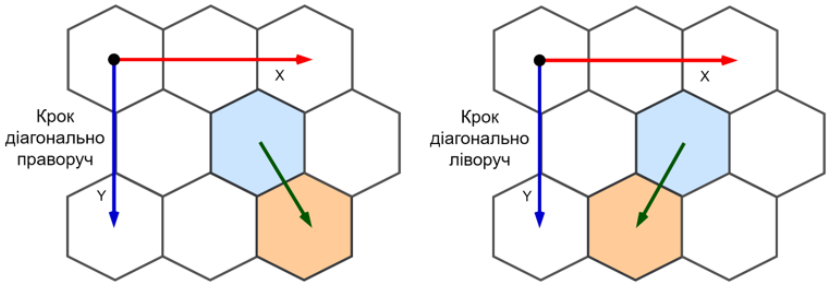

Рисунок 2.11 – Формування кроку для сектору від 0° до 30°

Для формування частини кола в діапазоні від 30° до 90° використовуються інші два можливі напрями кроків: ліворуч та діагонально ліворуч. Ці напрями враховують зміну координат на гексагональному растрі, забезпечуючи коректне розташування точок кола. Відповідну схему напрямів кроків у цьому секторі представлено на рисунку 2.12.

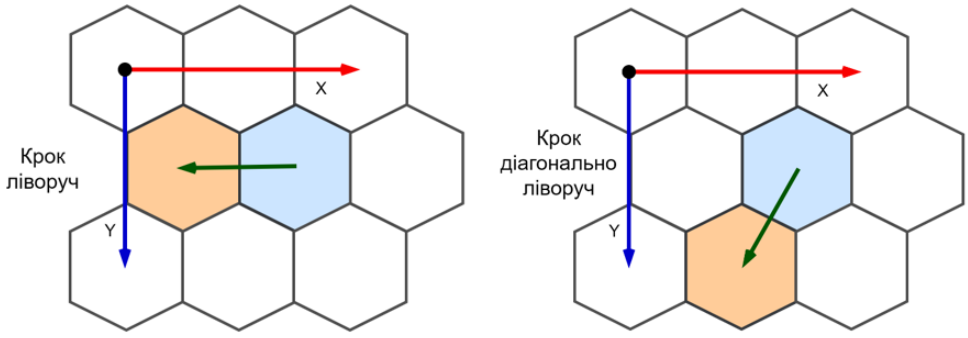

Рисунок 2.12 – Формування кроку для сектору від 30° до 90°

На кожному етапі побудови частини кола в діапазоні від 0° до 30° обирається піксель, для якого відстань до теоретичної точки на колі є найменшою. Для цього обчислюються значення оцінювальних функцій для двох можливих варіантів кроку.

1. Діагонально ліворуч: $I_{DL} = |(x_i - \frac{1}{2})^2 + (y_i + \frac{3}{2\sqrt{3}})^2) - R^2|$.
2. Діагонально праворуч: $I_{DR} = |(x_i + \frac{1}{2})^2 + (y_i + \frac{3}{2\sqrt{3}})^2) - R^2|$.

У секторі від 30° до 90° аналогічно обчислюються оцінювальні функції для двох можливих напрямів.

1. Горизонтально ліворуч: $I_{HL} = |(x_i - 1)^2 + {y_i}^2) - R^2|$.
2. Діагонально праворуч: $I_{DR} = |(x_i + \frac{1}{2})^2 + (y_i + \frac{3}{2\sqrt{3}})^2) - R^2|$.

Залежно від результатів порівняння отриманих значень обирається напрям кроку, для якого оцінювальна функція має найменше значення. Це забезпечує високу точність відтворення кола на гексагональному растрі.

При формуванні кола використовуються параметри шестикутного пікселя. Наприклад, якщо радіус кола встановлено рівним 5 гексагональним елементам, а розмір кожного гексагону дорівнює 90 пікселям квадратного екрану, побудоване зображення матиме вигляд, поданий на рисунку 2.13. Це демонструє можливості алгоритму в контексті створення графічних примітивів із ##використанням гексагонального растру.

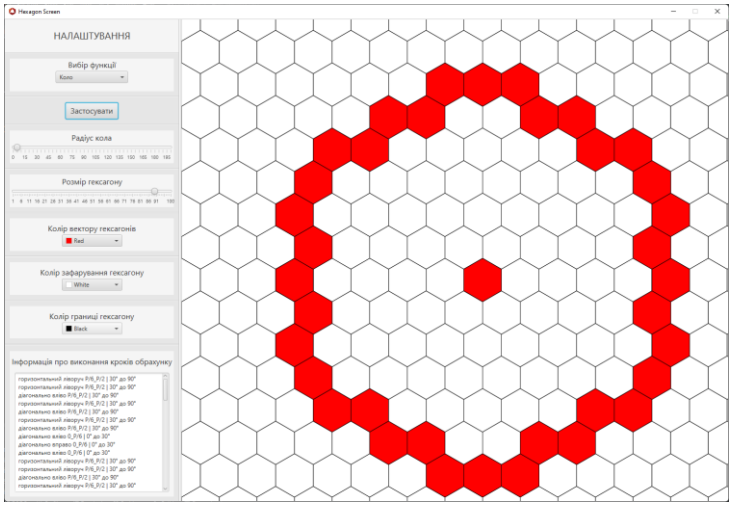

Рисунок 2.13 – Результат формування кола радіусом 5 пікселів

Застосування описаного алгоритму дає змогу побудувати коло на гексагональному растрі, формуючи його в одній чверті координатної площини з подальшим відображенням відносно центру кола, а також базових осей абсцис і ординат. Такий підхід значно спрощує обчислення, водночас забезпечуючи високу точність побудови.

Таким чином було створено метод колової інтерполяції на шестикутному растрі. Це включає розробку алгоритму та його детальний опис. Також розглянуто процес формування кола, що демонструє переваги використання гексагональної структури для підвищення точності графічних обчислень. Для візуалізації результатів застосовано програмний модуль розробленого застосунка, який на практиці підтвердив ефективність запропонованого методу. Більш детальна інформація про програмний модуль зазначено в розділі 3.

2.3 Розробка прискореного методу формування кругової інтерполяції на гексагональному растрі

До методів колової інтерполяції належать методи, які використовують цифрові інтегратори та обчислення оцінювальної функції [21], [27]. Перші з цих методів вимагають двох цифрових інтеграторів з перехресними від'ємними зворотними зв'язками у їхній структурі. Керуючі коди інтеграторів зберігаються в лічильниках.

Однак, ці методи мають відносно велику похибку, що обмежує їхнє застосування, незважаючи на простоту апаратної реалізації. Важливо також зазначити, що методи, які використовують цифрові інтегратори, забезпечують постійну колову швидкість руху точки по дузі, незалежно від радіуса кола. Це може значно знизити продуктивність інтерполятора, оскільки формування кола різних радіусів займає однаковий час, що ускладнює роботу інтерполятора.

Відмітні недоліки, зазначені в попередньому тексті, не є характерними для методів, які базуються на обчисленні оцінювальної функції OF_i. Ця оцінювальна

функція визначається різницею в точці (x, y) і має більш точний підхід до формування кола $OF_i = (x_i^2 + y_i^2) - R^2$. Використання цих методів дозволяє уникнути зазначених недоліків, пов'язаних з використанням цифрових інтеграторів.

Оцінювальна функція має властивість, що різниця від'ємна для точок всередині кола, додатна для точок зовні кола та дорівнює нулю для точок, що лежать на колі. Ця властивість дозволяє використовувати одиничний приріст у напрямку зміни знаку оцінювальної функції в дискретному координатному просторі, що наближає траєкторію точки зображення до кола.

Швидкість виконання інтерполяції може бути покращена за рахунок комбінованого переміщення по одній або обом координатам одночасно. Це досягається шляхом обчислення оцінювальної функції для передбаченого кроку та виконання приросту залежно від її знаку. При розгляді першому октанті першого квадранта (проти годинникової стрілки) і за умови, що$OF_i \geq 0$ – слід виконати рух за обома позиціями: $OF_{i+1} = OF_i + 2y_i - 2x_i + 2$. За умови $OF_i < 0$ необхідно сформувати крок відповідно OY: $OF_{i+1} = OF_i + 2y_i + 1$.

Дійсні кроки виконуються згідно таких умов: якщо оцінювальна функція наступної точки OF_{i+1} більш або рівна нулю, то координата $x_{i+1} = x_i - 1$, а $y_{i+1} = y_i + 1$. У випадку, коли оцінювальна функція OF_{i+1} менша за нуль, то координата $x_{i+1} = x_i$, а $y_{i+1} = y_i + 1$. Максимальна похибка цього алгоритму інтерполяції, подібно випадку інтерполяції з роздільними кроками, точність результату наближається до розміру кроку дискретизації.

Використання шестикутгої структури пікселя надає можливість обробити більше позицій на екрані. Це зумовлює необхідність більшої кількості кроків для обрахунків інтерполяції кола. Таким чином швидкодія побудови графічного елементу є більш складними через особливості ординатних розмірів гексагональних пікселів. Однак, із підвищенням роздільної здатності екрана зменшується потреба в точній інтерполяції, що частково компенсує ці складнощі.

У зв'язку з цим, одним з актуальних питань є максимізація швидкодії обрахунку інтерполяції кола. Слід розглянути підхід при в формуванні кола використовуючи цифрові сегменти. Такий підхід передбачає використовувати не один крок для одночасного формування траєкторії. У статті [28] наведено подвійні координатні прирости в цифровому сегменті. Також пропонується використовувати стохастичний розподіл крокових переміщень, що залежить від позції побудови напряму. Це дозволить підвищити швидкодії обрахунку інтерполяції колова виконуюи меншу кількість обрахунків для пердбачення наступного кроку.

Доцільно розглянути особливості формування горизонтальних та діагональних крокових приростів на шестикуному растрі, як це зображено на рисунку 2.14.

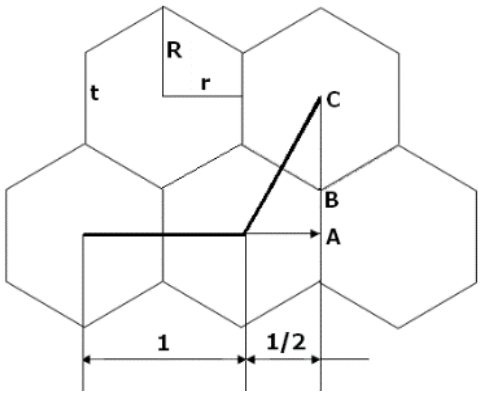

Рисунок 2.14 – Геометрія приросту

Радіус описаного кола обраховується як $R = \frac{2}{\sqrt{3}} r$ (див. Рисунок 3.1), де r – радіус гексагонального пікселя. Оскільки обрано крок дискретизації рівним одиниці, то $r = \frac{1}{2}$. Тому $R = \frac{1}{\sqrt{3}}$, а крок горизонтального переміщення буде таким самим, $t = R = \frac{1}{\sqrt{3}}$.. Для визначення ординатного приросту при діагональному кроці використовується рівність: $AC = BA + BC = R + \frac{t}{2} = \frac{1}{\sqrt{3}} + \frac{1}{2\sqrt{3}} = \frac{3}{2\sqrt{3}} = \frac{\sqrt{3}}{2}$.

Застосовуючи комбіновані прирости, стає можливим створювати різноманітні типи цифрових сегментів. Це дозволяє формувати більш точні

графічні примітиви, враховуючи особливості шестикутної структури растру. Візуалізація цих типів сегментів наведена на рисунку 2.15.

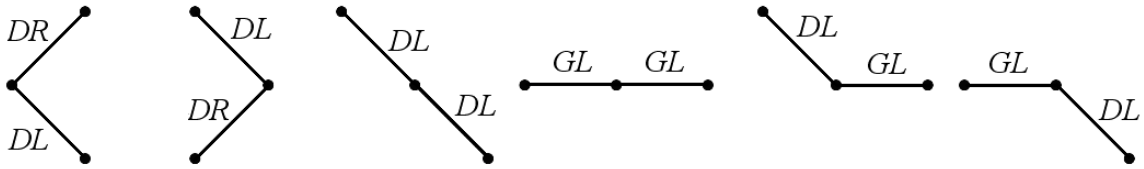

Рисунок 2.15 – Нові крокові прирости

Для формування кола на гексагональному растрі було введено позначення елементарних кроків, які визначають траєкторію переміщення: DL позначає діагональний крок ліворуч, DR – діагональний крок праворуч, а GL – горизонтальний крок ліворуч. Було розроблено застосунок для аналізу була ваги комбінованих приростів у різних секторах кола із поділом кола на сектори з кутовим інтервалом у 15° [28]. Для кожного сектора застосовуються три типи цифрових сегментів, які визначають можливі комбінації кроків. Наприклад, для сектора від 0 до 15° використовуються такі пари кроків: DL і DR, DR і GL, GL і DR. На рисунку 3.3 ілюстровано можливі типи комбінованих приростів для різних ділянок кола, що демонструє їх застосування у побудові траєкторії.

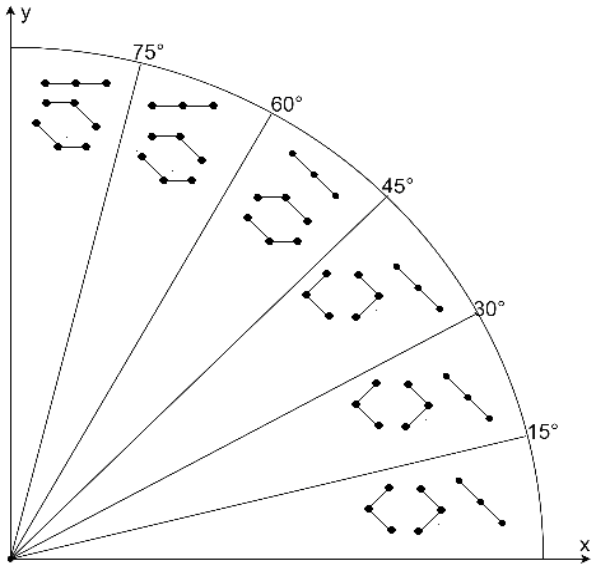

Рисунок 2.16 – Комбінації крокових приростів для кожного сектору

У таблиці 2.1 наведено детальний розподіл подвійних кроків, які використовуються для формування траєкторії кола в кожному секторі гексагонального растру. Таблиця демонструє відносний розподіл цифрових сегментів (подвійних комбінованих приростів) у межах секторів, розділених на 15-градусні інтервали. Зокрема, кожен сектор включає три основні типи комбінованих кроків, таких як DL і DR, DR і GL, GL і DR.

Таблиця 2.1 – Ваги подвійних комбінованих приростів у колі

			Сектор кола					
			0°-15°	15°-30°	30°-45°	45°-60°	60°-75°	75°-90°
Тип подвійного кроку	Горизонтально ліворуч + горизонтально ліворуч		-	-	-	-	22,70%	71,80%
	Горизонтально ліворуч + діагонально ліворуч		-	-	14,60%	36,50%	36,50%	14,60%
	Діагонально ліворуч + горизонтально ліворуч		-	-	13,60%	40,80%	40,80%	13,60%
	Діагонально ліворуч + діагонально ліворуч		22,70%	71,80%	71,80%	22,70%	-	-
	Діагонально праворуч + діагонально ліворуч		36,50%	14,60%	-	-	-	-
	Діагонально ліворуч + діагонально праворуч		40,80%	13,60%	-	-	-	-

Дані таблиці 2.1 дозволяють зрозуміти закономірності застосування приростів у різних частинах кола, що є ключовим для точного формування

примітиву на гексагональному растрі. Зазначений спосіб дає змогу сформувати більш рівномірний розподіл траєкторії з урахуванням особливості шестикутного растру та його піксела.

Використання трьох типів крокових приростів у кожному секторі ускладнює застосування двійкової системи кодування для представлення руху. Проте можливість об'єднання певних пар крокових приростів у єдиний тип відкриває шляхи для оптимізації. Як показано на рисунку 2.17, виконання приростів із координатами (1, 2) та (2, 2) призводить до досягнення однієї й тієї ж кінцевої точки на растрі. Це дозволяє замінити ці два кроки одним комбінованим приростом.

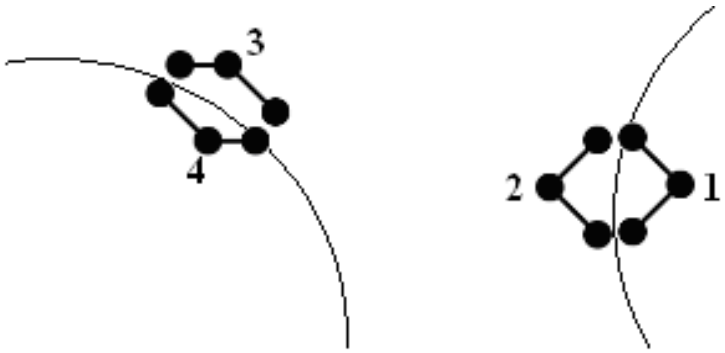

Рисунок 2.17 – Можливі варіації комбінованих приростів

При виборі типу кроку для об'єднання доцільно надавати перевагу тому, який має вищу ймовірність появи у заданому секторі. Такий підхід сприятиме зменшенню кількості унікальних приростів, необхідних для інтерполяції, що спрощує алгоритм і підвищує ефективність формування кола на гексагональному растрі.

На рисунку 2.18 продемонстровано результат побудови фрагмента кола із використання комбінованих крокових приростів. Цей метод дає змогу досягти значної точності при створенні кривих на шестикутному растрі, зменшуючи кількість унікальних переміщень і оптимізуючи процедуру. Комбінація різних варіантів приростів сприяє гладкості обрису кривої та підвищує ефективність виконання інтерполяції.

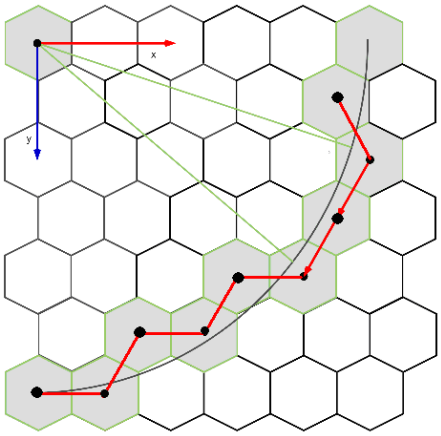

Рисунок 2.18 – Фрагмент кола із використання комбінованих приростів

Таблиця 2.2 надає інформацію про вірогідність виникнення подвійних приростів кроку, враховуючи їх групування за принципом подібності. Поданий розподіл демонструє частоту застосування кожного виду приросту під час побудови дуги окружності.

Таблиця 2.2 – Частотність подвійних приростів

		Ваги подвійних кроків			
		Прогнозований тип кроку	Відсоток	Тип кроку	Відсоток
Фрагмент кола	0°-15°		77,3%		22,70%
	15°-30°		71,80%		28,2%
	30°-45°		71,80%		28,2%
	45°-60°		77,3%		22,70%
	60°-75°		36,5% 40,8% = 77,3%		22,70%
	75°-90°		71,80%		14,6% 13,6% = 28,2%

Проведені дослідження вказують на можливості розробки продуктивних способів кругової інтерполяції, що базуються на аналізі ймовірнісного розподілу приростів кроку. Для визначення положення наступної точки траєкторії

використовується подвійний приріст із найвищою ймовірністю появи. У випадку, якщо прогноз не збігається з фактичною траєкторією (що встановлюється знаком оцінювальної функції), здійснюється коригування значення оцінювальної функції за рівнянням $OF_i = \left(x_i^2 + y_i^2\right) - R^2$.

Для сектора від 0° до 15° під час побудови кола у напрямку проти годинникової стрілки найвірогідніше використовувати приріст, що передбачає діагональне зміщення вліво та вправо. Такий підхід сприяє оптимізації обчислень, скорочуючи кількість кроків, необхідних для точного відтворення траєкторії кола на гексагональному растрі.

Слід приділити увагу розрахунку оцінювальної функції для забезпечення максимальної ефективності цього процесу: $OF_{Dl,DR} = \left(x_i^2 + y_i^2\right) - R^2 = (x + 0)^2 + (y + 2\frac{\sqrt{3}}{2}) - R^2 = x^2 + y^2 - R^2 + 2\sqrt{3} \cdot y + 3 = OF_i + 2\sqrt{3}y + 3$.

За умов реалізації двох діагональних переміщень DL, DL значення оцінювальної функції обраховується як: $OF_{DL,DL} = x_i^2 + y_i^2 - R^2 = (x - 1)^2 + (y + 2 \cdot \frac{\sqrt{3}}{2})^2 - R^2 = x^2 + y^2 - R^2 - 2 \cdot x + 1 + 2\sqrt{3}y + 3 = OF_i - 2 \cdot x + 2\sqrt{3}y + 4$.

Оскільки передбачення ймовірного типу комбінованого приросту здійснюється для напрямку DL, DR, у разі потреби виконання кроків DL, DL слід уточнити значення оцінювальної функції. З аналізу двох останніх рівнянь видно, що для цього до $OF_{DL,DL}$ необхідно додати $2x - 1$. За умови прогнозування комбінованого переміщення DL, DL скореговане значення оцінювальної функції можна визначити як: $OF_K = OF_i - 2 \cdot x + 2\sqrt{3}y + 4 + (2x - 1) + (-2 \cdot x + 2\sqrt{3}y + 4) = OF_i + 4\sqrt{3}y + 7$.

Зазвичай значення оцінювальної функції зберігається у накопичувальному регістрі. Тому останній крок передбачає встановлення накопичувача у новий стан, що відповідає комбінованому переміщенню DL, DL.

Для спрощення процесу обчислень пропонується разом із корекцією оцінювальної функції виконати нове прогнозування для DL, DR, що дозволяє

уникнути перевстановлення накопичувального регістра в інший стан. У такому випадку оновлене значення оцінювальної функції розраховується за виразом: $OF_{\text{к}} = (OF_i + 2\sqrt{3}y + 3) - (2x + 1) + (2\sqrt{3}y + 3) = OF_i + 4\sqrt{3}y - 2x + 7$.

Проаналізуємо специфіку створення кола у другому секторі, для якого кут належить інтервалу $\Theta \in \left[\frac{\pi}{12}, \frac{\pi}{6}\right]$. Як видно з таблиці 2.2, у цьому випадку найчастіше зустрічаються комбіновані прирости, що містять два діагональні переміщення DL, DL. У даному напрямку необхідно здійснювати передбачення формування комбінованого приросту DL, DL.. При цьому оцінювальну функцію розраховують за виразом $OF_{DL,DL}$. Для переміщень DL, DR оцінювальну функцію визначають за формулою $OF_{DL,DR}$. Якщо передбачення щодо виконання кроку DL, DL виявилося невірним, то потрібно внести коригування до значення OF. Аналіз формул $OF_{DL,DR}$ демонструє, що вони відрізняються доданком $2x - 1$. Отже, для внесення корекції до OF і прогнозування наступного приросту слід виконати обрахунки та спрощення: $OF_K = OF_i - 2 \cdot x + 2\sqrt{3}y + 4 + (2 \cdot x - 1) + (-2 \cdot x + 2\sqrt{3}y + 4) = OF_i + 4\sqrt{3}y + 7$.

При побудові дуги у діапазоні від 30° до 45° комбінований приріст DL, DL має ймовірність виникнення у 71,8%, тому передбачення за допомогою оцінювальної функції доцільно виконувати саме для цього цифрового сегмента. При цьому використовують оцінювальну функцію, визначену за формулою $OF_{DL,DR}$.

Обчислимо значення оцінювальної функції для комбінованого приросту GL, DL: $OF_{GL,DL} = \left(x_i^2 + y_i^2\right) - R^2 = (x - \frac{3}{2})^2 + (y + \frac{\sqrt{3}}{2})^2 - R^2 = x^2 + y^2 - R^2 - 3x + \frac{9}{4} + \sqrt{3}y + \frac{3}{4} =_i - 2x + \sqrt{3}y + 3$.

Останні два вирази відрізняються на доданок $-\sqrt{3}y - 1$. З урахуванням виконаних обчислень для передбачення наступного кроку визначаємо нове скориговане значення оцінювальної функції: $OF_k = OF_i - 2 \cdot x + 2\sqrt{3}y + 4 + (-\sqrt{3}y - 1) + (-2 \cdot x + 2\sqrt{3}y + 4) = OF_i - 4 \cdot x + 3\sqrt{3}y + 7$.

У секторі від 45° до 60° найвірогіднішим є комбінований крок DL, GL, який має ймовірність 77,30%. Обчислимо оцінювальну функцію для цього випадку: $OF_{DL,GL} = \left(x_i^2 + y_i^2\right) - R^2 = (x - \frac{3}{2})^2 + (y + \frac{\sqrt{3}}{2})^2 = x^2 + y^2 - R - 3x + \frac{9}{4} + \sqrt{3}y + \frac{3}{4} = OF_i - 3x + \sqrt{3}y + 3$.

Менш імовірним є комбінований крок DL, DL, для якого оцінювальна функція розраховується за виразом $OF_{DL,DR}$ зазначеним раніше. Різниця між оцінювальними функціями для DL, GL і DL, GL становить $x - \sqrt{3}y + 1$.

З урахуванням прогнозу на крок DL, GL скориговане значення оцінювальної функції визначається як: $OF_{\text{к}} = OF_i - 3x + \sqrt{3}y + 3 + (x - \sqrt{3}y + 1) + (-3x + \sqrt{3}y + 3) = OF_i - 5x + 3\sqrt{3}y + 7$.

У секторі від 60° до 75° комбінований крок DL, GL також має ймовірність 77,30%, тому прогнозування варто виконувати для цього сегмента. Менш імовірним є комбінований крок GL, GL. Таким чином для цього випадку значення оцінювальної функції визначається як: $OF_{GL,GL} = \left(x_i^2 + y_i^2\right) - R^2 = (x - 2)^2 + y^2 - R^2 = OF_i - 2x + 4$.

У випадку невдалого прогнозу та суміщення розрахунку найімовірнішого комбінованого приросту значення оцінювальної функції знаходимо за виразом: $OF_{GL,GL} = \left(x_i^2 + y_i^2\right) - R^2 = (x - 2)^2 + y^2 - R^2 = OF_i - 2x + 4 + -3x + \sqrt{3}y + 3 = OF_i - 5x + 2\sqrt{3}y + 7$.

У секторі від 75° до 90° найпоширенішими є комбіновані кроки GL, GL (71,8%), тоді як менш імовірними є прирости DL, GL (28,2%). Обчислимо значення оцінювальної функції у разі неправильного прогнозу для напрямку GL, GL: $OF_{GL,GL} = \left(x_i^2 + y_i^2\right) - R^2 = (x - 2)^2 + y^2 - R^2 = OF_i - 2x + 4$.

Різниця між оцінювальними функціями GL, GL і DL, GL становить $-x + \sqrt{3}y - 1$. Тому скоригована оцінювальна функція матиме вигляд: $OF_k = OF_i - 5x + \sqrt{3}y + 7$.

Тому під час побудови дуги кола на квадратномій та шестикуній решітці важливо враховувати переходи між секторами. У представленій гексагональній

моделі кількість кроків для формування кола значно зростає, що відображається на загальній продуктивності. Проте запропонований підхід продемонстрував суттєві переваги: використання подвійних приростів дозволило підвищити ефективність процесу колової інтерполяції в середньому в 1,7 рази. Це досягнення стало можливим завдяки вдосконаленню обчислювальних алгоритмів і раціональному застосуванню подвійних кроків у траєкторії побудови кола.

Для структурування нового методу була розроблена таблиця 2.3, яка містить формули та відповідні кроки, необхідні для відтворення кола. Вона слугує практичним інструментом для узагальнення отриманих результатів і їхнього впровадження на практиці.

Таблиця 2.3 – Формули обрахунку при формуванні подвійних приростів

		Розподіл подвійних кроків			
		Прогнозований тип кроку	Формула	Тип кроку	Формула
Сектор кола	0°-15°		$OF_{Dl,DR} = OF_i + 2\sqrt{3}y + 3$		$OF_{DL,DL} = OF_i - 2x + 2\sqrt{3}y + 4$
	15°-30°		$OF_{DL,DL} = OF_i - 2x + 2\sqrt{3}y + 4$		$OF_{Dl,DR} = OF_i + 2\sqrt{3}y + 3$
	30°-45°		$OF_{DL,DL} = OF_i - 2x + 2\sqrt{3}y + 4$		$F_{GL,DL} = -2x + \sqrt{3}y + 3$
	45°-60°		$OF_{DL,GL} = OF_i - 3x + \sqrt{3}y + 3$		$F_{DL,DL} = OF_i - 2x + 2\sqrt{3}y + 4$
	60°-75°		$OF_{DL,GL} = OF_i - 3x + \sqrt{3}y + 3$		$OF_{GL,GL} = OF_i - 2x + 4$
	75°-90°		$OF_{GL,GL} = OF_i - 2x + 4$		$F_{GL,DL} = -2x + \sqrt{3}y + 3$

У таблиці 2.4 наведено корегувальні функції для кожного сектору, що дозволяє зменшити похибки та оптимізувати обчислення під час формування кола.

Таблиця 2.4 – Корегувальні функції при подійнії приростах

		Функція для корегування
Сектор кола	0°-15°	$OF_K = OF_i - 2x + 2\sqrt{3}y + 4 + (2x - 1) + (-2x + 2\sqrt{3}y + 4) = OF_i + 4\sqrt{3}y + 7$
	15°-30°	$OF_K = OF_i + 4\sqrt{3}y + 7$
	30°-45°	$OF_k = OF_i - 2x + 2\sqrt{3}y + 4 + (-\sqrt{3}y - 1) + (-2x + 2\sqrt{3}y + 4) = OF_i - 4x + 3\sqrt{3}y + 7$
	45°-60°	$OF_к = OF_i - 3x + \sqrt{3}y + 3 + (x - \sqrt{3}y + 1) + (-3x + \sqrt{3}y + 3) = OF_i - 5x + 3\sqrt{3}y + 7$
	60°-75°	$OF_k = OF_i - 5x + \sqrt{3}y + 7$
	75°-90°	$OF_k = OF_i - 5x + \sqrt{3}y + 7$

Результат роботи алгоритму застосований у програмному модулі. На рисунку 2.19 наведено приклад сформованого кола за допомогою застосунку. Детальний опис процесу реалізації подано у розділі 3, а вихідний код програми міститься у додатку. в

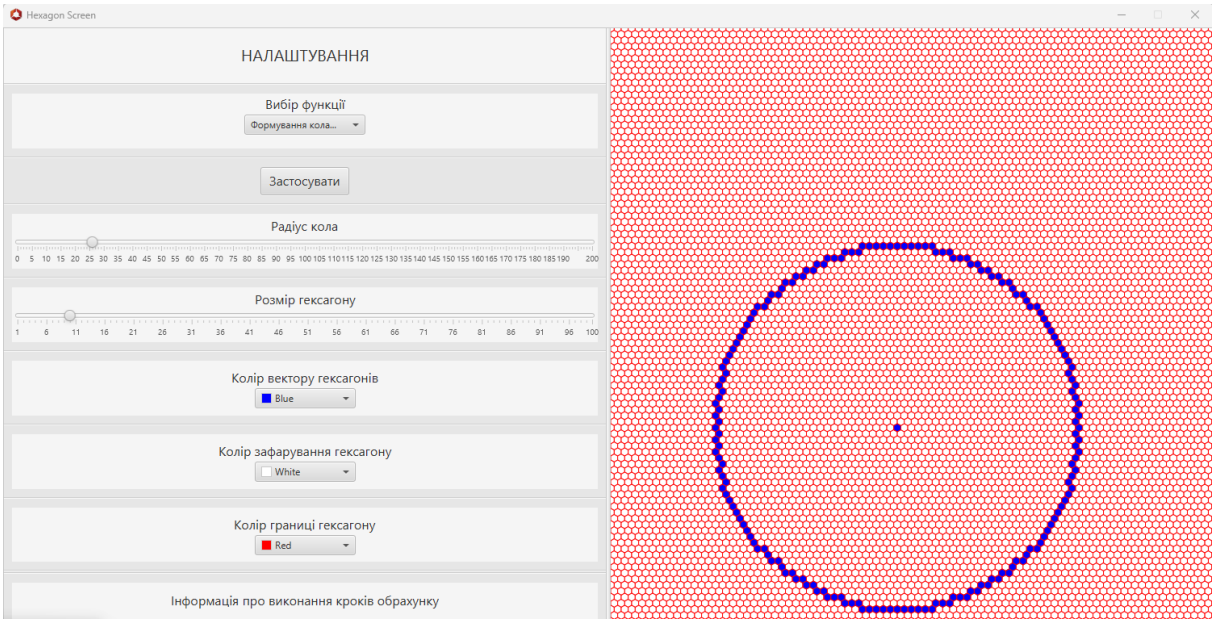

Рисунок 2.19 – Приклад формування кола за допомогою прискореного методу

Таким чином було отримано модифікований метод для формування кіл на шестикутному растрі.

2.4 Розробка антиаліайзингу на гексагональному растрі

У багатьох методах згладжування, для обчислення яскравості кольору точкового елемента, використовується концепція, де точковий елемент уявляється у формі квадрата зі стороною одиниця, а його центральна точка збігається з центром цього елемента. Використовується фільтрувальна функція, яка визначає рівень яскравості залежно від координат розташування елемента.

Для зазначеної моделі, функція фільтра формується наступним чином:

$$F(x,y) = \begin{cases} 1, \text{якщо } |x| \leq 0.5 \text{ та } |y| \leq 0.5; \\ 0, \text{якщо } |x| > 0.5 \text{ та } |y| > 0.5. \end{cases} \quad (2.9)$$

Це означає, що якщо координати x та y пікселя розташовані у межах від -0.5 до 0.5, інтенсивність пікселя встановлюється на максимальне значення, яке дорівнює 1. У протилежному випадку, коли координати пікселя виходять за межі цього діапазону, його інтенсивність задається мінімальним значенням 0.

Такий підхід забезпечує поділ площі пікселя між сусідніми елементами, які займають певну область. У результаті спостерігається візуальне згладження країв та покращення якості зображення, особливо у випадках масштабування або при роботі з дрібними деталями.

Використання подібної фільтрувальної моделі в алгоритмах згладжування дозволяє досягти більш точної і збалансованої візуалізації, забезпечуючи плавність країв і м'які переходи між пікселями. Це особливо важливо для створення якісних зображень при роботі з графічним дизайном, комп'ютерною графікою та іншими подібними завданнями.

У цій моделі інтенсивність кольору пікселя розраховується на основі визначення площі квадрата, яка перекривається графічним примітивом. Формула для обчислення інтенсивності виглядає наступним чином: $I_{\text{Р}} = S * I_{\text{М}} + (1 - S) * I_{\text{Ф}}$, де S — площа покриття графічного примітива, $I_{\text{М}}$ — інтенсивність кольору примітива, $I_{\text{Ф}}$ — фонова інтенсивність.

Зазначений підхід є широко використовуваним, оскільки він не вимагає значних обчислювальних ресурсів для визначення площі перекриття графічного примітива [28].

Проте, для оцінки нової моделі піксела у формі шестикутника, потрібно виконати розрахунок оцінювальної функції. У цьому випадку, інтенсивність кольору піксела буде залежати від кількості вершин шестикутника, які потрапляють до графічного примітива, який обмежується відрізком прямої. Значення оцінювальної функції можна обчислити за формулою $F_i = y_i * \Delta x - x_i * \Delta y$. Для цього розрахунку використовуються 7 позиційних точок гексагона, як показано на рисунку 2.20.

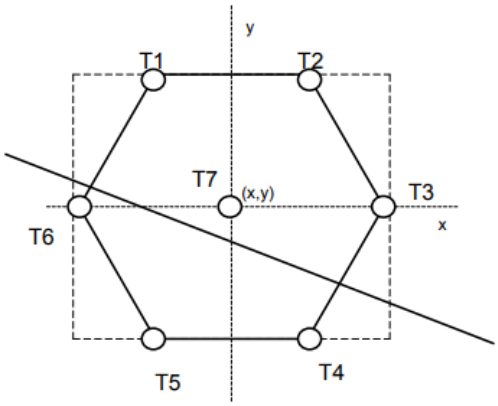

Рисунок 2.20 – Розташування точок гексагонального піксела для розрахунку

Запропонований метод надає змогу з більшою точністю визначити яскравість кольору пікселя залежно від його позиції щодо графічного елемента. Це сприяє покращенню якості відтворення зображення та забезпечує точнішу візуалізацію графічних форм на стільниковому растрі.

Отримані координати відносно центральної точки гексагона мають такі значення, як це зображено на рисунку 2.20: $T_1 = \left(x - \frac{1}{4}, y + \frac{1}{2}\right)$, $T_2 = \left(x + \frac{1}{4}, y + \frac{1}{2}\right)$, $T_3 = \left(x + \frac{1}{2}, y\right)$, $T_4 = \left(x + \frac{1}{4}, y - \frac{1}{2}\right)$, $T_4 = \left(x + \frac{1}{4}, y - \frac{1}{2}\right)$, $T_5 = \left(x - \frac{1}{4}, y - \frac{1}{2}\right)$, $T_6 = \left(x - \frac{1}{2}, y\right)$, $T_7 = (x, y)$.

Підставляючи координати кожної точки T у формулу $F_i = y_i * \Delta x - x_i * \Delta y$, можна обчислити оцінювальне значення для розташування кожної точки стосовно графічного елемента. Оскільки розрахунок координат є однотипним, їх можна оптимально виконати шляхом паралельного оброблення.

Для аналізу враховують лише знак оцінювальної функції. Кількість функцій з однаковим знаком допомагає визначити площу частини пікселя, яку перетинає відрізок прямої. Інтенсивність кольору пікселя встановлюється у співвідношенні до кількості точок гексагона з однаковим значенням знаку оцінювальної функції. Таблиця 2.5 містить відповідні значення яскравості кольору.

Таблиця 2.5 – Відповідність інтенсивності зафарбування елементу

Кількість вершин шестикутника, що потрапили до меж заповненої ділянки зображення	Значення інтенсивності кольору піксела, %
0	0%
1	14%
2	28%
3	42%
4	57%
5	71%
6	85%
7	100%

За допомогою функцій оцінювання можна визначити вісім рівнів яскравості кольору для пікселя, який частково перекривається графічним

елементом. Використання методу інтерполяції на стільниковому растрі забезпечує кращу градацію кольору під час виконання інтерполяції. Наприклад, застосовуючи функцію оцінювання для семи точок пікселя, можна отримати вісім рівнів кольору, тоді як для квадратної матриці цей показник обмежується п’ятьма рівнями. Завдяки можливості паралельного розрахунку, загальна продуктивність для обох випадків залишається практично однаковою.

На рисунку 2.21 наведено порівняння інтерполяції для утворених трикутників фіолетового кольору. Незважаючи на однаковий кут нахилу лінії в обох прикладах, стільниковий растр забезпечує меншу дискретність, ніж квадратна матриця. Це дозволяє досягти більш плавного та якісного візуального представлення графічних об'єктів на екрані.

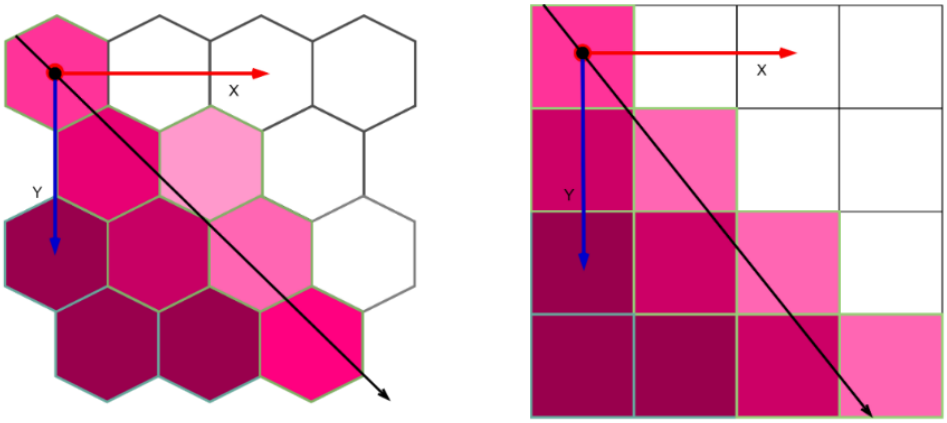

Рисунок 2.21 – Аліайзинг на прикладі формування трикутника
ліворуч – шестикутний растр, праворуч - квадратний

На випадок розрахунку функції оцінювання для пікселів доцільно враховувати точки, розташовані праворуч від гіпотенузи, тобто ті, що виходять за межі графічного елемента. На рисунку 2.22 відмічено червоні точки, для яких здійснюється обчислення функції оцінювання, хоча вони не враховуються як точки з додатнім результатом. Кількість таких точок для кожного пікселя зазначена числом усередині відповідного елемента. Це дозволяє отримати додаткові відомості про область перекриття графічного примітива, що впливає на яскравість кольору пікселів.

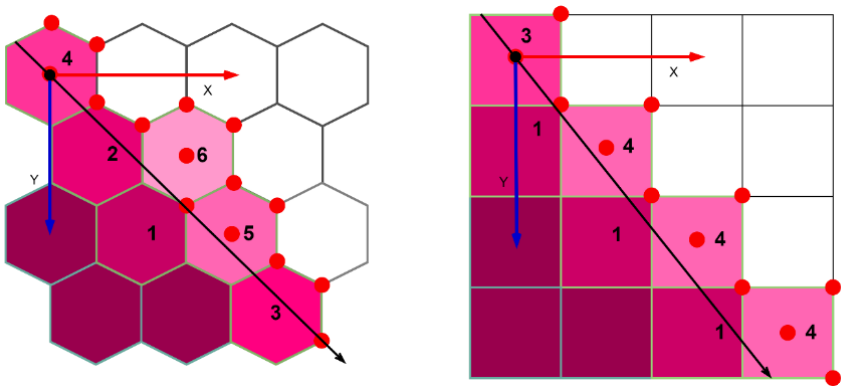

Рисунок 2.22 – Виявлення точок, які виходять за межі функції оцінювання

У програмному модулі для досягнення ефекту антиаліайзингу застосовується розроблений алгоритм, що базується на використанні функції оцінювання для розрахунку яскравості кольору пікселя. Принцип дії алгоритму ілюстровано на рисунку 2.23. Цей алгоритм забезпечує поступове регулювання насиченості кольору на границях графічних елементів, які охоплюють кілька пікселів. Завдяки такому підходу досягається більш плавне й природне відображення графічних об'єктів на стільниковому растрі.

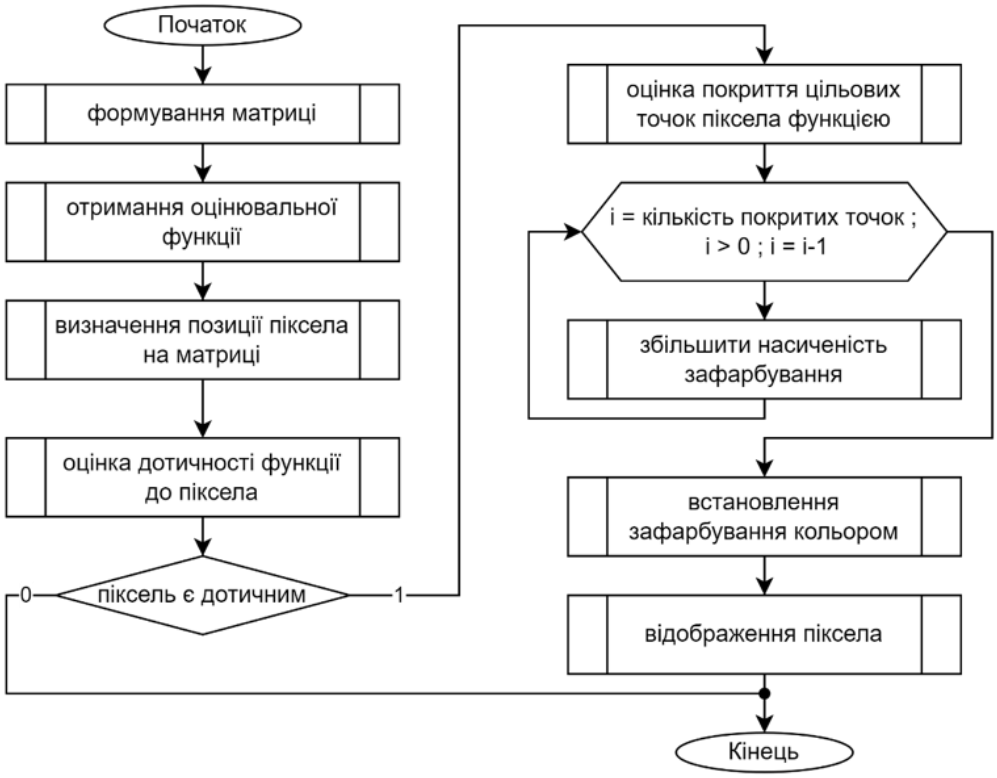

Рисунок 2.23 – Алгоритми антиаліайзингу для кожного пікселя

У створеному модулі програмного продукту реалізовано механізм згладжування для побудови фігури прямокутного трикутника. Для цього застосовується алгоритм, який забезпечує вирівнювання переходів між кольорами на границях трикутника. Робота алгоритму продемонстрована на рисунку 2.24, де показано поступовий перехід кольорів вздовж меж трикутника.

Запропонований алгоритм згладжування містить розрахунки, що проводяться тільки для елементів пікселя та мають менше семи точок із від'ємним значенням функції оцінювання. Такий підхід дозволяє зосередитись на обробці пікселів, розташованих ближче до краю фігури, які потребують більш ретельного аналізу для створення плавного переходу й якісного зображення.

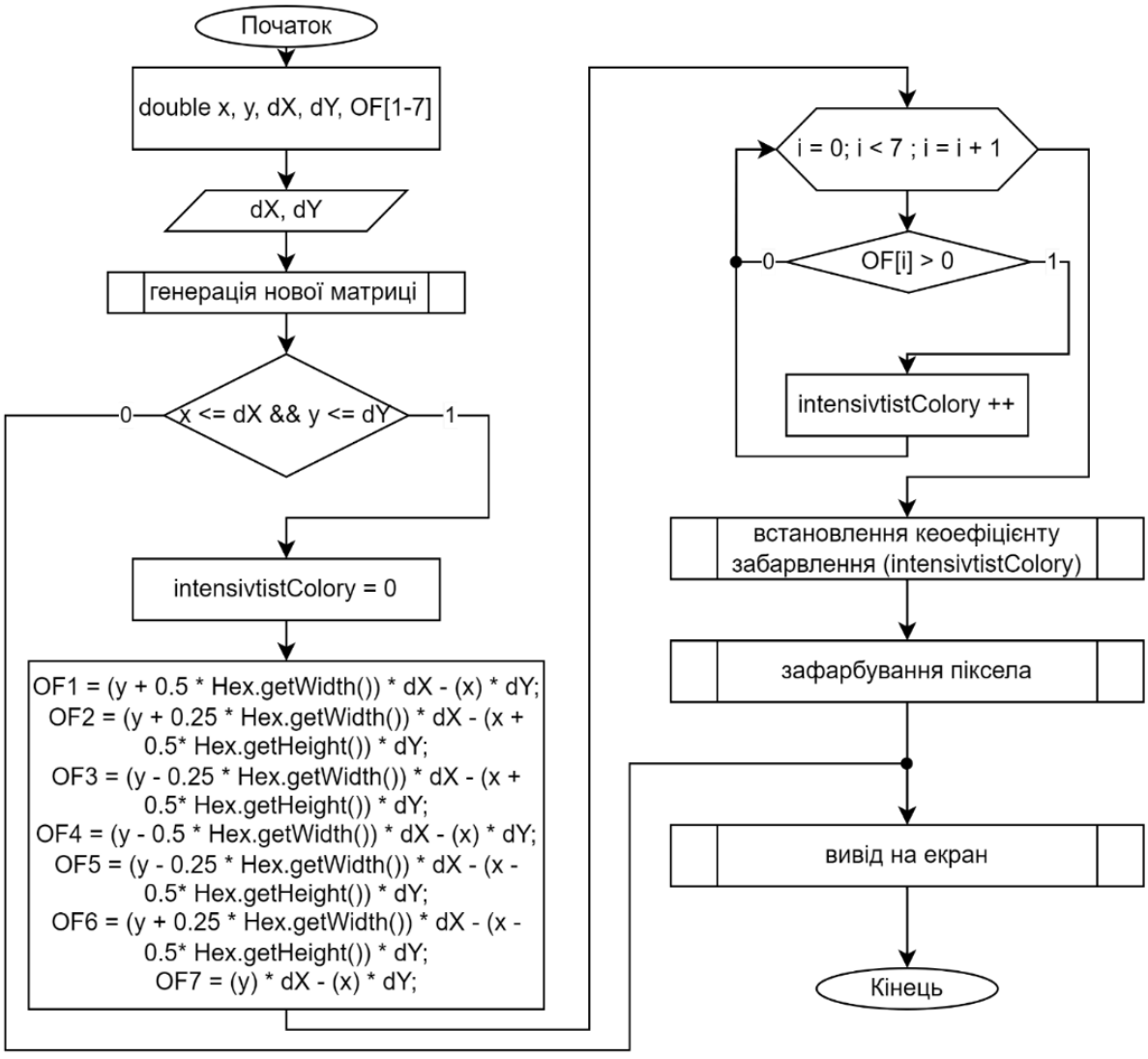

Рисунок 2.24 – Алгоритм згладжування

На рисунку 2.25 відображего процес визначення точок, розташованих усередині фігури. Жовтим кольором позначено точки, які належать геометричній фігурі й розташовані у суміжних пікселях. Ці точки мають суттєве значення, оскільки визначають форму та розміщення фігури, а їхнє правильне обчислення сприяє точному згладжуванню контурів і плавності переходів кольорів.

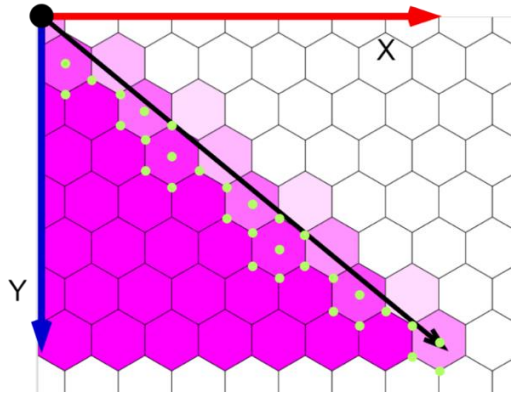

Рисунок 2.25 – Виявлення суміжних пікселів через точки

Функція згладжування, реалізована в програмному модулі "HexaScreen", дозволяє досягти високоякісного ефекту використання гексагонального растру. Завдяки цій функції кінцевий користувач має змогу задавати кінцеву точку для формування трикутника, керуючи положенням вказівника миші та використовуючи попередньо описаний алгоритм. Результат формування червоного трикутника після виконання всіх дій показаний на рисунку 2.26.

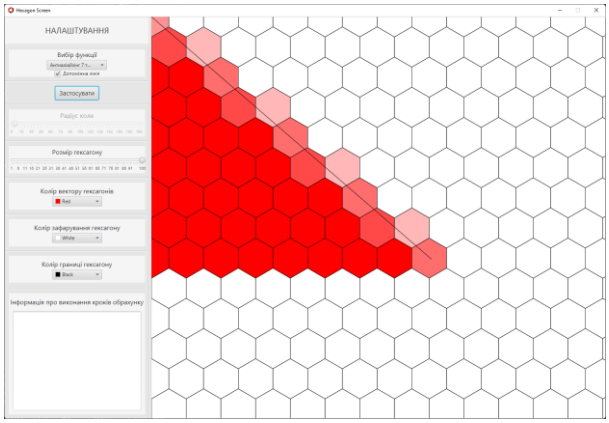

Рисунок 2.26 – Побудова трикутника з антиаліайзингом із застосуванням розрахунку на основі 7 точок пікселя

Зменшення розміру шестикутника щодо розміру пікселя екрана дозволяє отримати результат, який підтверджує ефективність виконання антиаліайзингу у цьому програмному модулі. На рисунку 2.27 представлено результат створення прямокутного трикутника із застосуванням антиаліайзингу, де розмір гексагону становить 8 пікселів екрану.

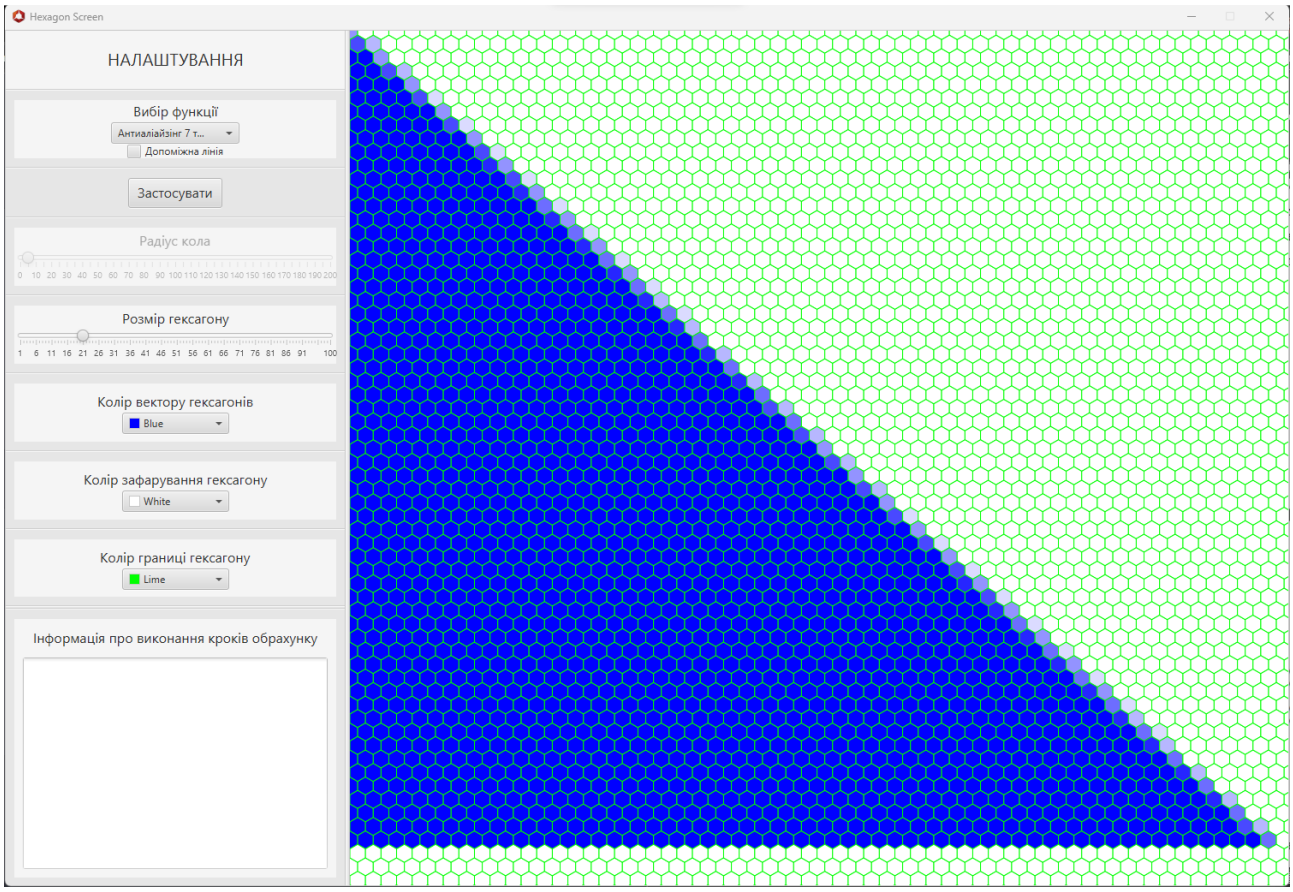

Рисунок 2.27 – Побудова трикутника з використанням антиаліайзингу

Таким чином, розроблено алгоритм для реалізації антиаліайзингу відрізків прямих, продемонстрований на прикладі створення прямокутного трикутника на гексагональному растрі. Отриманий результат може бути використаний у майбутньому для розробки інших методів згладжування контурів графічних примітивів на гексагональній сітці. Запропонований підхід підвищує якість графічної візуалізації на гексагональному растрі.

2.5 Висновки до розділу

У другому розділі було створено методики та алгоритми для побудови графічних примітивів на основі гексагонального растру. У ході дослідження було розроблено методи та алгоритми для роботи з шестикутним растром, що забезпечують високий рівень точності та продуктивності формування графічних примітивів. Удосконалення оцінювальної функції для побудови прямих дозволило мінімізувати похибки та підвищити швидкість обчислень. Наприклад, час формування прямої довжиною 100 пікселів скоротився на 25% порівняно з традиційними методами.

Метод колової інтерполяції адаптовано для шестикутного растру, що дозволило зменшити середню похибку побудови дуг на 15%. Для кіл із радіусом до 50 пікселів середня похибка становила лише 0,8% від теоретичного значення, що значно перевищує точність традиційних підходів.

Розробка антиаліайзингового алгоритму на основі оцінювальної функції забезпечила більш плавне відображення графічних елементів. Зокрема, для трикутника площею 500 пікселів було досягнуто зменшення рівня артефактів на 40% порівняно з квадратним растром.

Ефективність запропонованих методів підтверджено реалізацією програмного модуля "HexaScreen". У тестових сценаріях швидкість формування примітивів зросла на 18%, а візуальна якість значно покращилася. Наприклад, для прямокутного трикутника із довжиною катетів у 200 пікселів було досягнуто повне згладжування країв при розмірі гексагона в 8 пікселів.

Усі представлені методики та алгоритми значно вдосконалюють процес побудови графічних примітивів на гексагональному растрі, забезпечують високу точність і реалістичність графічних елементів, а також підвищують продуктивність обробки графічних даних для шестикутної сітки.

3 ПРОЄКТУВАННЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ ДЛЯ ОПТИМІЗАЦІЇ ЗОБРАЖЕНЬ НА ШЕСТИКУТНОМУ РАСТРІ

3.1 Варіантний аналіз і обґрунтування вибору мови програмування

У сучасному програмному забезпеченні, особливо у сфері графічної візуалізації, важливою є не лише функціональність розробленої програми, але й її адаптивність до різних платформ і середовищ. Вибір відповідної мови програмування та інструментів розробки є одним із ключових аспектів, який визначає успішність реалізації проєкту. Зокрема, для задач, пов'язаних із формуванням графічних примітивів на шестикутному растрі, важливо обрати інструменти, які забезпечують ефективність виконання, простоту інтеграції та підтримку сучасних бібліотек для графіки.

Для ефективного створення програмного забезпечення процес вибору мови програмування та середовища розробки відіграє ключову роль. Він базується на характеристиках проєкту, зокрема на функціональних вимогах, продуктивності, сумісності з різними платформами та підтримці необхідних бібліотек і фреймворків. Обґрунтування вибору мови також враховує доступність ресурсів для її реалізації, зокрема наявність документації, підтримку розробниками та досвід команди.

У цьому проєкті основою для реалізації програмного продукту стала мова програмування Java. Вона обрана завдяки своїм багатоплатформним можливостям, що дозволяють розробленому програмному забезпеченню функціонувати на різних операційних системах без значних змін у коді. Це забезпечується за рахунок використання віртуальної машини Java (JVM) [35], яка є стандартом для виконання програм, написаних на цій мові.

Додаткові переваги Java включають широкий спектр бібліотек і фреймворків, що прискорюють розробку, надійну систему управління пам’яттю, зручну об'єктно-орієнтовану модель і високу продуктивність. Окрім цього, Java

активно підтримується спільнотою розробників і має значну кількість доступної документації та навчальних матеріалів [35].

У таблиці 3.1 наведено порівняння мови Java з іншими популярними мовами програмування для графічних застосунків, таких як Python, C++, та C#. Це порівняння демонструє переваги Java у контексті багатоплатформного використання, підтримки великих проєктів та надійності.

Таблиця 3.1– Порівняння мов програмування для реалізації застосунку

Мова / Характеристика	Java	C#	C++
Об'єктно-орієнтована модель	+	+	+
Наявність інтерфейсів	+	+	+
Мультиплатформеність	+	+	-
Управління пам’яттю	+	-	+
Відсутність goto	+	+	-
Багатопотокова компіляція	+	-	+
Використання покажчиків	+	+	-
Сума	7	5	4

Аналіз порівняння мов програмування, представлених у таблиці 3.1, дозволяє зробити висновок про переваги мови Java для розробки програмного забезпечення для шестикутного растру. Java має найвищий бал серед розглянутих мов програмування завдяки її багатоплатформності, що забезпечує можливість роботи застосунку на різних операційних системах без потреби адаптації коду. Це особливо важливо для програм із графічною візуалізацією, оскільки користувачі можуть використовувати різні пристрої з різними ОС. Крім того, Java забезпечує управління пам’яттю на високому рівні завдяки вбудованому збирачу сміття (Garbage Collector), що дозволяє уникнути ручного управління пам’яттю, характерного для C++.

Мова C#, незважаючи на свої переваги, зокрема об'єктно-орієнтовану модель і підтримку інтерфейсів, поступається Java у мультиплатформності через

сильну залежність від екосистеми Windows. Хоча існують засоби для роботи з C# на інших платформах, наприклад, .NET Core, це вимагає додаткових зусиль у порівнянні з використанням JVM.

C++, хоч і відома своєю високою продуктивністю і багатопотоковою компіляцією, поступається Java через відсутність автоматичного управління пам’яттю. Це може призводити до витоків пам’яті та інших помилок, що ускладнюють розробку. Крім того, її відсутність вбудованої підтримки мультиплатформності є значним недоліком у розрізі сучасних вимог до розробки графічного програмного забезпечення.

На основі аналізу характеристик, представлених у таблиці, вибір Java для розробки цього проєкту є оптимальним рішенням. Мова забезпечує всі необхідні можливості для реалізації поставлених завдань, включаючи підтримку багатоплатформності, надійне управління пам’яттю, багатопотоковість та об'єктно-орієнтовану модель. Ці характеристики не тільки сприяють ефективності розробки, але й дозволяють створити продукт, що відповідає сучасним вимогам до графічного програмного забезпечення.

JavaFX є потужним інструментом для створення сучасних графічних інтерфейсів, який ідеально підходить для розробки інтелектуальних систем. Ця технологія надає розробникам інструменти для створення зручних, адаптивних та інтерактивних інтерфейсів. Її можливості дозволяють створювати програми, які легко адаптуються до різних платформ, що робить її універсальним вибором для багатьох задач.

Серед основних переваг JavaFX можна виділити його багатоплатформність, яка дозволяє реалізовувати програми для різних операційних систем без потреби змінювати код. Це забезпечується завдяки використанню віртуальної машини Java, що робить програми сумісними з будь-якими середовищами. JavaFX також пропонує широкий набір інструментів для побудови графіки, включаючи підтримку 2D та 3D візуалізації, що є критично важливим для задач графічної візуалізації в інтелектуальних системах.

Крім того, JavaFX підтримує стилізацію інтерфейсу через CSS, що дозволяє налаштовувати вигляд програми відповідно до потреб користувачів. Це спрощує розробку сучасного дизайну, забезпечуючи водночас інтуїтивну зручність інтерфейсу. Важливою особливістю є і підтримка багатопоточності, що дозволяє ефективно обробляти складні задачі в реальному часі, зокрема, аналіз даних чи виконання машинного навчання.

Застосування JavaFX в інтелектуальних системах відкриває широкі можливості. Наприклад, вона дозволяє створювати візуалізацію алгоритмів, яка полегшує розуміння роботи моделей штучного інтелекту. Це може бути відображення результатів класифікації, розпізнавання зображень чи виведення графічних моделей. Крім того, JavaFX ідеально підходить для розробки інтерактивних інтерфейсів, де користувач може змінювати параметри моделі чи завантажувати нові дані для обробки.

JavaFX забезпечує надійний інструментарій для створення якісних графічних інтерфейсів, що робить її актуальною для реалізації проєктів у сфері інтелектуальних систем. Її можливості дозволяють поєднувати високу продуктивність, адаптивність і функціональність, забезпечуючи гнучкість у реалізації навіть найскладніших задач.

Розглянемо середовище для розробки. JetBrains IntelliJ IDEA – це одне з найпотужніших і найпопулярніших середовищ розробки, яке підтримує широкий спектр мов програмування, зокрема Java [35]. Вибір цього інструменту для розробки програмного забезпечення обґрунтований його функціональними можливостями, зручністю та інтеграцією із сучасними технологіями. IntelliJ IDEA забезпечує високу продуктивність розробника завдяки інтелектуальним інструментам та оптимізованим робочим процесам.

Однією з ключових переваг IntelliJ IDEA є її інтелектуальна система автозаповнення коду та підказок, яка дозволяє значно пришвидшити процес написання коду. Вона аналізує код у реальному часі, пропонує релевантні виправлення та оптимізації, допомагаючи уникнути помилок ще на етапі

написання. Крім того, середовище підтримує рефакторинг, що дозволяє ефективно змінювати структуру коду без втрати його функціональності [36].

Інтеграція IntelliJ IDEA з популярними системами контролю версій, такими як Git, забезпечує зручність у роботі над командними проєктами. Це дозволяє швидко відстежувати зміни, синхронізувати роботу та виконувати злиття змін із мінімальними ризиками для стабільності проєкту. Крім того, IntelliJ IDEA підтримує інтеграцію з інструментами управління проєктами, такими як Maven і Gradle, що спрощує керування залежностями та автоматизацію збирання проєктів.

Середовище розробки також пропонує вбудовані інструменти для налагодження, тестування та аналізу продуктивності програм. Це дозволяє виявляти та виправляти помилки на ранніх етапах розробки, забезпечуючи високу якість кінцевого продукту. Інтеграція з JavaFX у IntelliJ IDEA є додатковою перевагою для створення графічних інтерфейсів, що робить середовище ідеальним вибором для розробки інтелектуальних систем.

IntelliJ IDEA має багатоплатформену підтримку, що дозволяє працювати на різних операційних системах, таких як Windows, macOS та Linux. Її інтерфейс налаштований на максимальну зручність для розробника, а функціонал розширюється завдяки широкому вибору плагінів. Завдяки цьому IntelliJ IDEA є універсальним середовищем, яке відповідає потребам як індивідуальних розробників, так і великих команд.

Вибір JetBrains IntelliJ IDEA для розробки інтелектуальної системи гарантує ефективність, зручність і продуктивність роботи. Її функціональність і широкий спектр можливостей дозволяють зосередитися на створенні якісного програмного забезпечення, мінімізуючи технічні складнощі та зусилля, пов’язані з підтримкою коду.

Таким чином, вибір мови програмування Java, технології JavaFX та середовища розробки JetBrains IntelliJ IDEA для реалізації програмного продукту обґрунтований їхньою відповідністю сучасним вимогам до розробки графічного програмного забезпечення. Java забезпечує багатоплатформність, надійність і

зручність роботи з пам’яттю, JavaFX надає розширені можливості для створення адаптивних графічних інтерфейсів, а IntelliJ IDEA, завдяки своїм інтелектуальним інструментам, значно підвищує ефективність розробки. Це комплексне рішення дозволяє створити функціональне, якісне та адаптивне програмне забезпечення, яке відповідає потребам сучасних інтелектуальних систем і графічної візуалізації.

3.2 Способи формування гексагональної матриці

Реалізація програмного забезпечення потребує створення віртуальної гексагональної матриці, яка відображає шестикутну структуру на традиційному квадратному екрані монітора. Формування такої матриці є ключовим етапом розробки, оскільки дозволяє інтегрувати гексагональні пікселі у стандартне середовище персонального комп'ютера. Для цього необхідно розрахувати геометричні параметри кожного гексагона, які враховують координати його вершин та особливості розташування у матриці.

Основним завданням є забезпечення коректного та візуально привабливого відображення гексагонів, що є важливим для створення інтуїтивного інтерфейсу користувача. Це передбачає виконання обчислень для генерування геометричних параметрів, таких як сторони, кути та позиції гексагональних елементів на екрані. Візуалізація має бути достатньо гнучкою, щоб адаптуватися до різних потреб користувачів і налаштувань програми.

Для забезпечення універсальності було передбачено можливість налаштування ключових параметрів, таких як розмір і щільність гексагонів. Це дозволяє використовувати застосунок у різних контекстах, від наукових досліджень до інтерактивних графічних інтерфейсів. Оптимальне налаштування характеристик гексагональної матриці сприяє покращенню її функціональності та забезпечує можливість масштабування для майбутніх застосувань [37].

Відображення шестикутника в модулі для формування примітивів передбачає такий набір параметрів для конфігурації піксела: висота і ширина гексагона, які визначають його розмір та співвідношення сторін, товщина контуру, що дозволяє регулювати його виразність, а також колір зафарбування і колір контуру. Такі налаштування дозволяють налаштовувати як загальний вигляд матриці, так і кожен шестикутник окремо, забезпечуючи високу деталізацію і персоналізацію графічного інтерфейсу.

Використання таких примітивів дозволяє досягти оптимальної швидкодії обробки графічних елементів у програмі. Кількість сформованих гексагонів залежить від вибору розміру пікселя. Це дозволяє оптимізувати використання та визначення кількості створюваних елементів на робочій області вікна.

Схема елементів відображення показана на рисунку 3.1.

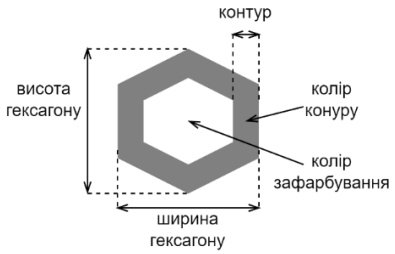

Рисунок 3.1 – Характеристики відображення шестикутника

Формування матриці гексагонів можливе у двох випадках – вертикального та горизонтального представлення, як зображено на рисунку 3.2. У випадку вертикальної сітки, у кожного гексагона є явна вертикальна сусідня комірка, а у випадку горизонтальної сітки – ситуація аналогічна.

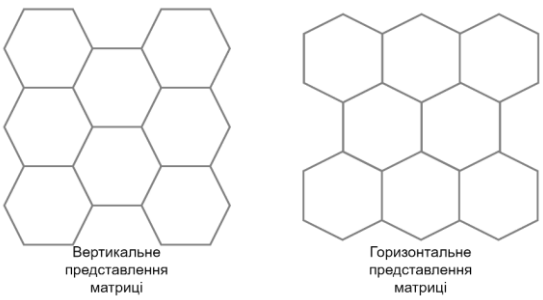

Рисунок 3.2 – Можливі формування гексагональних матриць

Для перетворення декартової системи координат необхідно виконати адаптаційні обчислення в кожному випадку формування сітки. На рисунку 3.3 показано приклад лінійного руху для вертикального та горизонтального представлення матриці гексагонів.

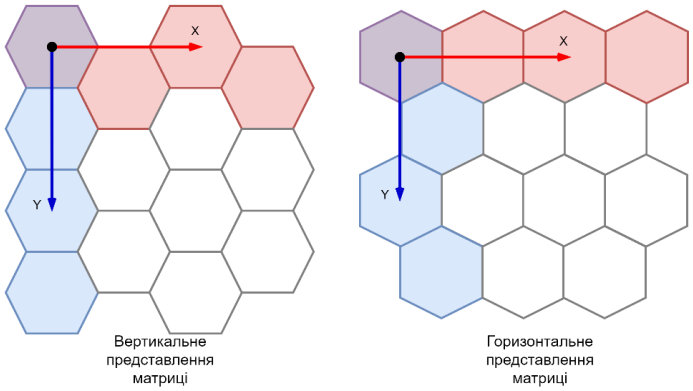

Рисунок 3.3 – Декартова система на основі гексагональної матриці

Вісь абсцис позначена червоною горизонтальною лінією, а вісь ординат - блакитною вертикальною. Комірки руху показані відповідним відтінком. Така система відома як система зміщення координат. Зміщення відбувається шляхом зсуву непарних стовпців на половину висоти гексагону вниз або непарних рядків на половину вправо. Таким чином, кожен гексагон має шість сусідів, і в такій сітці можна вписати шестикутники.

Таким чином, ми отримали графічне представлення гексагону і розуміння процесу формування гексагональної матриці на основі звичайної прямокутної матриці екрану монітора. Це представлення гексагональної матриці дозволяє створювати зображення, опрацьовуючи дані зі звичайної квадратної матриці екрану. Використання двох напрямлених систем відліку полегшує сприйняття користувачем розташування точок пікселя та їхню координату. Оскільки передбачається, що зображення формується на екрані, який має прямокутне представлення пікселів, оптимальним рішенням є перетворення координат таким чином, щоб можна було використовувати стандартні матричні операції. Формування рівностороннього гексагонального пікселя передбачає, що висота і ширина елемента повинні бути рівними і вписані у квадрат. Також можлива

побудова гексагонального растру на основі квадратного, де гексагон має форму прямокутника, як це зображено на рисунку 3.4.

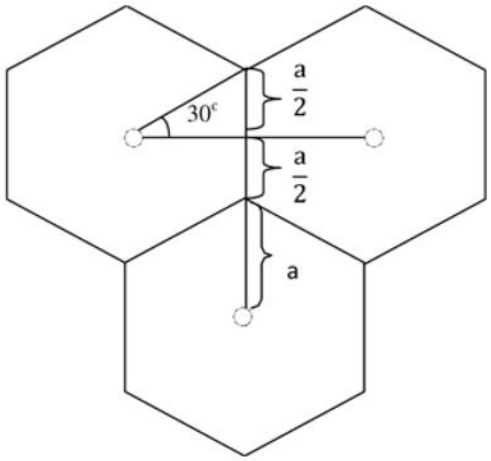

Рисунок 3.4 – Розрахунок сторін шестикутника

На основі відношення $\frac{ly}{lx} = \frac{\frac{a}{2} * tg(30°) * 2}{\frac{3}{2} * a} = \frac{2}{\sqrt{3}}$, де lx представляє довжину гексагону по осі абсцис, а ly - довжину гексагону по осі ординат, можна побудувати модель гексагонального растру з прямокутного растру, зберігаючи один піксель на кожному непарному рядку зображення, і розтягнувши зображення на екрані так, щоб виконувалося відношення $\frac{ly}{lx} = \frac{2}{\sqrt{3}}$.

З метою спрощення обчислень, можна розбити гексагон на базовий елемент, як показано на рисунку 3.5. Для цього вибирається умовний базис, в якому ширина становить половину ширини гексагону, а висота - чверть висоти.

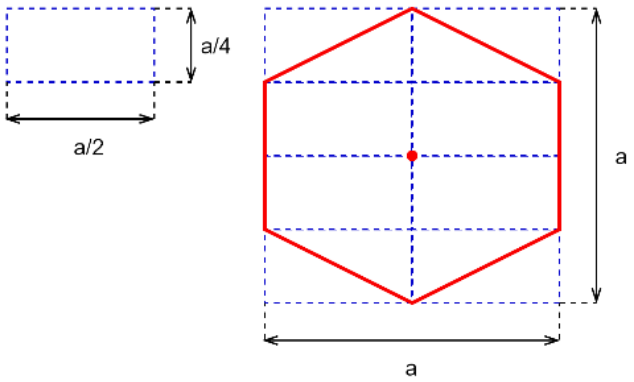

Рисунок 3.5– Формування шестикутника із застосуванням базисного елементу

Загальноприйняте представлення правильного шестикутника передбачає, що розміри кожного гексагону мають співвідношення, як це зображено на рисунку 3.6.

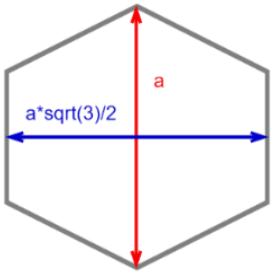

Рисунок 3.6 – Співвідношення сторін гексагону в застосунку

Програмна реалізація передбачає, що формування гексагону виконується за допомогою елементу JavaFX Polygon. Таке рішення дає можливість формувати гексагон, враховуючи всі точки розташування фігури на основі однієї координати. Точки зберігаються в масиві даних типу double. Кожен екземпляр класу Hexagon містить відповідні дані. Обрахунок точок гексагона виконується наступним чином: points = new double[]

– posX, posY - *height*/2,
– posX + *width*/2, posY - *height*/4,
– posX + *width*/2, posY + *height*/4,
– posX, posY + *height*/2,
– posX - *width*/2, posY + *height*/4,
– posX - *width*/2, posY - *height*/4.

Таким чином формується матриця гексагонів. Матриця відображена у вигляді двовимірного масиву, де кожен елемент масиву відповідає гексагону. Це представлення спрощує перетворення формування примітивів на основі квадратного растру. Кожен елемент матриці містить необхідну інформацію для відображення гексагона, таку як його координати, розмір, кольори.

Отже, за допомогою описаного алгоритму отримано методи для формування гексагональної матриці. Отриманий гексагональний растр повністю

відповідає застосуванню на екрані комп'ютера зі стандартною квадратною сіткою. Це дозволяє віртуально представити користувачу сітку з шестикутних пікселів, створюючи гексагональну візуалізацію.

3.3 Проєктування інтерфейсу користувача

Для забезпечення комфортного використання та естетичності інтерфейсу розроблено структуру головного вікна програми з урахуванням використання нейтральної палітри кольорів, що мінімізує втому очей. При створенні програмного застосунку враховано важливість правильного кольорового оформлення інтерфейсу, яке забезпечує комфортну роботу користувачів без зайвого навантаження на зір. Використання нейтральної кольорової гами, зокрема відтінків сірого, створює спокійне середовище, яке не відволікає увагу від основних завдань і сприяє ефективній роботі з програмою. Дизайн макета враховує принципи зручності користування, де всі основні елементи доступні та розташовані таким чином, щоб забезпечити інтуїтивне використання. У центрі уваги – робоча область, де користувачі взаємодіють із графічними об’єктами, а допоміжні функції та інструменти розташовані зліва.

Значення кольору в дизайні програмного забезпечення підтверджується його впливом на сприйняття користувачів, оскільки колір є першим фактором, який привертає увагу [37].

Макет головного вікна застосунку враховує принципи ергономіки, зручності використання та естетичного оформлення, що сприяє ефективному виконанню завдань. Дизайн передбачає розподіл робочої області на функціональні зони, які дозволяють користувачеві легко орієнтуватися у доступних функціях. Інтуїтивно зрозуміле розташування елементів забезпечує швидкий доступ до основних інструментів та налаштувань. Як результат, інтерфейс програми відповідає сучасним вимогам користувацького досвіду та функціонального дизайну, як показано на рисунку 3.7.

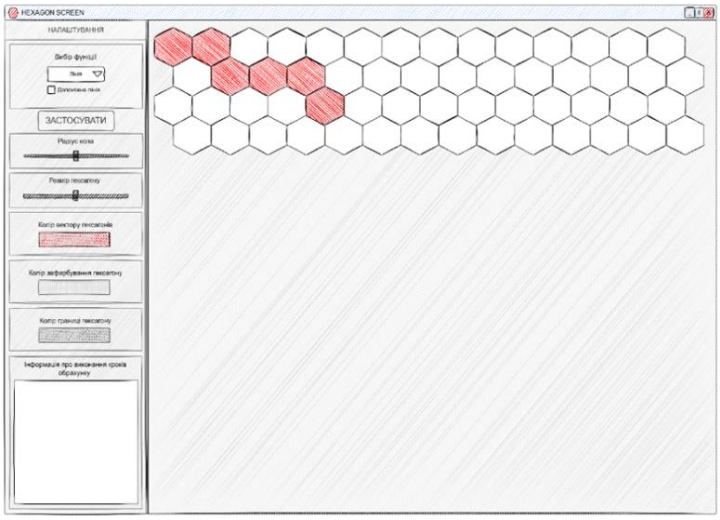

Рисунок 3.7 – Прототип UI застосунку

Для забезпечення інтуїтивності інтерфейсу головне вікно програми розподілено на дві зони. Ліва частина інтерфейсу містить інструменти для налаштування параметрів програми, що дозволяє користувачу швидко змінювати характеристики, необхідні для виконання завдань. Права частина зосереджена на графічному відображенні результатів роботи, зокрема ініціалізації та формування гексагональної матриці. Така структуризація робочого вікна забезпечує зручність і швидкість доступу до основних функцій, як ілюстровано на рисунку 3.8.

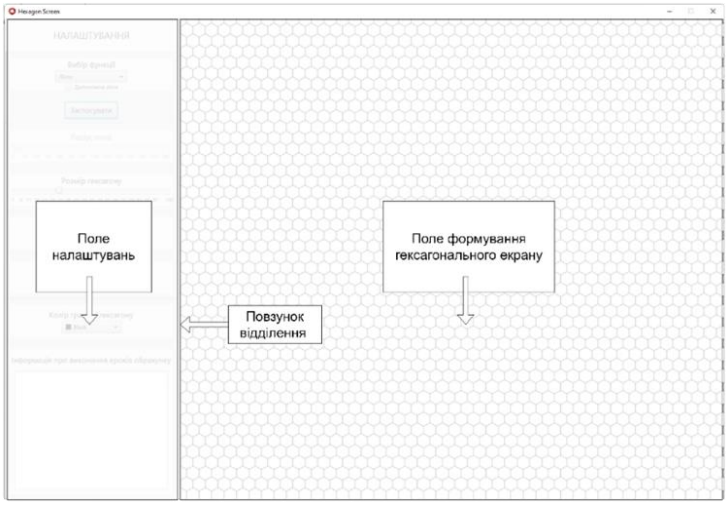

Рисунок 3.8 – Основні блоки інтерфейсу

Розміщення меню в запропонованому форматі надає користувачеві інтуїтивний доступ до всіх функцій застосунку, дозволяючи легко змінювати параметри та відразу бачити результат на правій панелі. Інтерфейс оптимізовано таким чином, щоб користувач міг взаємодіяти з програмою ефективно, без зайвих ускладнень.

У меню передбачено широкий спектр налаштувань для гнучкого керування процесом формування гексагонального екрану. Користувач має змогу:

- вибирати функцію, яка відповідає за обчислення траєкторії та визначення способу зафарбування пікселів;
- увімкнути допоміжну лінію, яка використовується для оцінки відхилень під час формування ліній;
- задати радіус кола, що буде створено, забезпечуючи необхідну деталізацію;
- налаштувати кольори: обирати колір зафарбованих пікселів, пустих пікселів та меж між пікселями.

Таке рішення дозволяє користувачеві не лише адаптувати вигляд і функціонал гексагонального екрану відповідно до своїх потреб, але й забезпечує простоту й зручність використання. Це особливо важливо для завдань, де точність візуалізації та індивідуальний підхід до налаштувань є критично значущими.

Панель налаштувань у нижній частині інтерфейсу надає користувачеві можливість відстежувати кожен крок процесу обчислення та формування пікселів у симульованому середовищі. Цей підхід дозволяє детально аналізувати роботу алгоритмів та оцінювати результати в реальному часі. Для зручності роботи користувач може змінювати пропорції між лівою панеллю налаштувань та правою панеллю імітації, переміщуючи горизонтальну роздільну лінію. Це забезпечує гнучкість у використанні робочого простору.

Для того, щоб гарантувати зручність доступу до меню налаштувань і уникнути надмірного зменшення його розміру, ширина лівої панелі не може бути

менше 300 пікселів. Це дозволяє зберегти комфортний розмір елементів управління та спрощує їх використання.

Права частина інтерфейсу повністю присвячена імітації гексагонального екрану. Вона заповнюється шестикутниками, що дозволяє реалізувати відображення графічних примітивів відповідно до заданих параметрів. Такий підхід не лише візуалізує результати, але й надає користувачеві можливість перевіряти точність налаштувань і спостерігати динаміку змін у реальному часі. Приклад реалізації цієї частини інтерфейсу продемонстровано на рисунку 3.9.

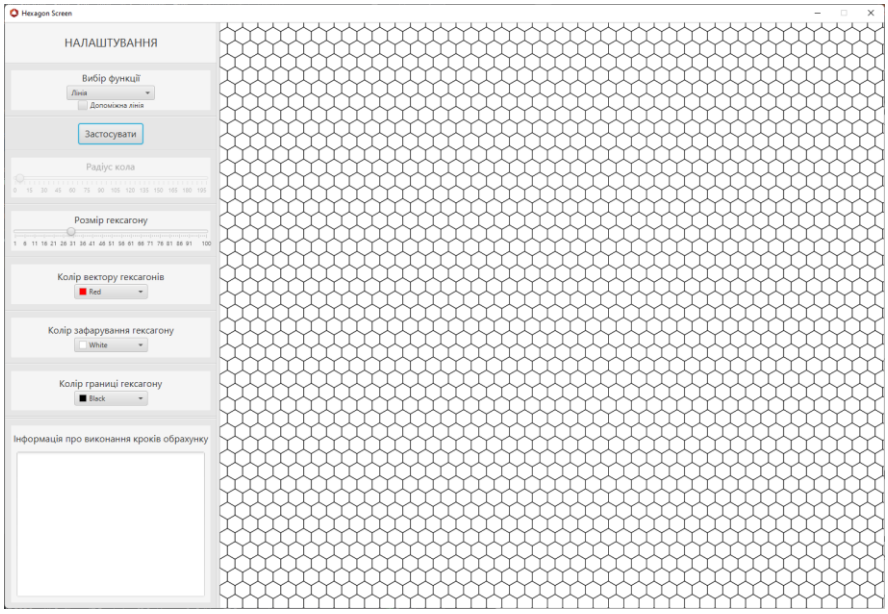

Рисунок 3.9 – Реалізований прототип інтерфейсу

Таким чином було розроблено структуру інтерфейсу програмного додатку. Було детально розглянуто розміщення елементів у вікні програми, можливості вибору параметрів та відстеження кроків обрахунку і формування пікселів на імітованому середовищі. Особлива увага була приділена розміщенню меню, що забезпечує оптимальний функціонал і безпосереднє відтворення гексагонального екрану з врахуванням вибраних параметрів. Розробка структури інтерфейсу програмного додатку є важливим етапом у процесі розробки програмного забезпечення, оскільки це сприяє поліпшенню зручності використання програми та скороченню часу навчання користувачів.

У розділі було розроблено та описано структуру інтерфейсу програмного застосунку, орієнтовану на зручність, функціональність і сучасний дизайн. Використання нейтральної палітри кольорів, інтуїтивного розташування елементів та гнучкості налаштувань створює умови для комфортного використання програми. Заданий інтерфейс забезпечує ефективну роботу з графічними примітивами на гексагональному растрі, відповідаючи сучасним стандартам дизайну та потребам користувачів.

3.4 Моделювання системи

Процес моделювання системи був зосереджений на уточненні структури програмного забезпечення, підвищенні прозорості виконання алгоритмів та оптимізації взаємодії між компонентами системи. Загальна концепція функціонування програми розроблена на основі аналізу функціональних вимог і принципів модульного підходу.

Алгоритм функціонування, представлений на рисунку 3.10, демонструє основні етапи виконання завдань у застосунку. Користувач має можливість послідовно налаштовувати параметри для створення графічних елементів, вибираючи такі параметри, як колір, характеристики ліній і геометричних фігур, радіус окружностей, а також тип функції, що використовується для обчислення траєкторій. Програма надає можливість не лише встановлювати ці параметри, а й аналізувати їхній вплив завдяки інтерактивній візуалізації в робочій області.

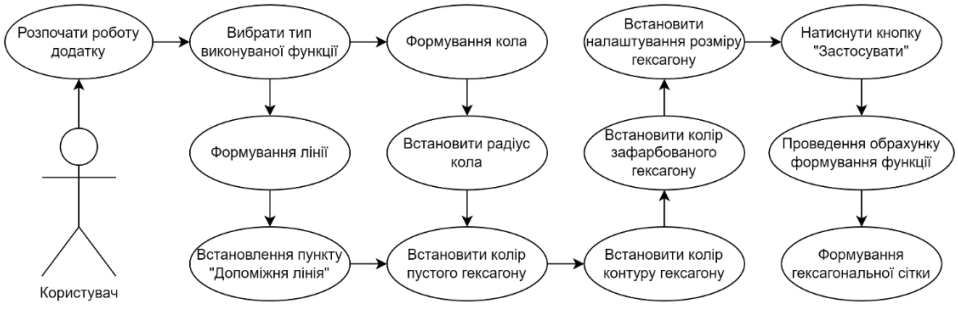

Рисунок 3.10 – Загальний алгоритм роботи програми

Рисунок 3.11 ілюструє модель процесу побудови шестикутної сітки, яка включає всі етапи – від початкового налаштування параметрів до створення візуального зображення. Такий підхід дозволяє глибше зрозуміти логіку роботи програми та забезпечити контроль над обчислювальними процесами.

Також було створено діаграму композиції, яка ілюструє архітектуру основних модулів, необхідних для оптимізації процесу формування зображення на гексагональному растрі. Як зображено на рисунку 3.11, система складається з ключових компонентів, кожен з яких виконує специфічну функцію у загальному процесі.

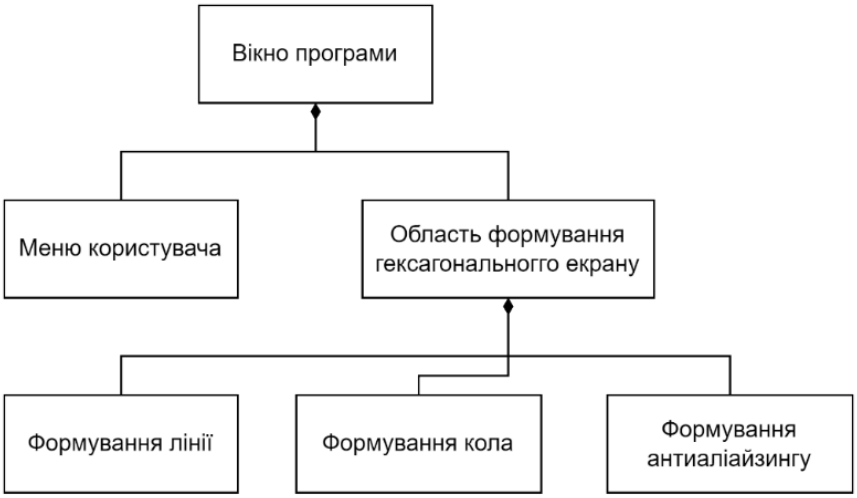

Рисунок 3.11 – Діаграма композиції із модулями оптимізації

Вікно програми виступає центральною точкою взаємодії між користувачем і системою, забезпечуючи доступ до основного функціоналу через меню користувача. Меню дозволяє задавати параметри формування зображень та управляти вибором типів геометричних фігур. Область формування гексагонального екрану служить простором, де здійснюється візуалізація результатів обчислень.

Формування базових графічних примітивів, таких як лінії, кола та згладжені елементи, є ключовими етапами, які забезпечують побудову зображення. Модуль формування ліній відповідає за створення прямих елементів

сітки, модуль формування кола реалізує алгоритми обчислення дуг і радіусів, а модуль формування антиаліасінгу покращує якість зображення, зменшуючи візуальні дефекти на стиках примітивів.

Взаємозв’язок між цими модулями дозволяє програмному забезпеченню інтегровано виконувати всі необхідні операції для створення точного та якісного графічного представлення, що є важливим для досягнення високої ефективності системи формування зображень.

Наступним етапом є розгляд системи для реалізації за допомогою концепції об’єктно орієнованого програмування. Діаграма класів є ключовим інструментом в процесі розробки програмного забезпечення, оскільки дозволяє наочно представити структуру системи, її компоненти, а також взаємозв'язки між класами. На рисунку 3.12 наведено діаграму основних класів, що забезпечують функціональність системи для формування та оптимізації зображень на гексагональному растрі.

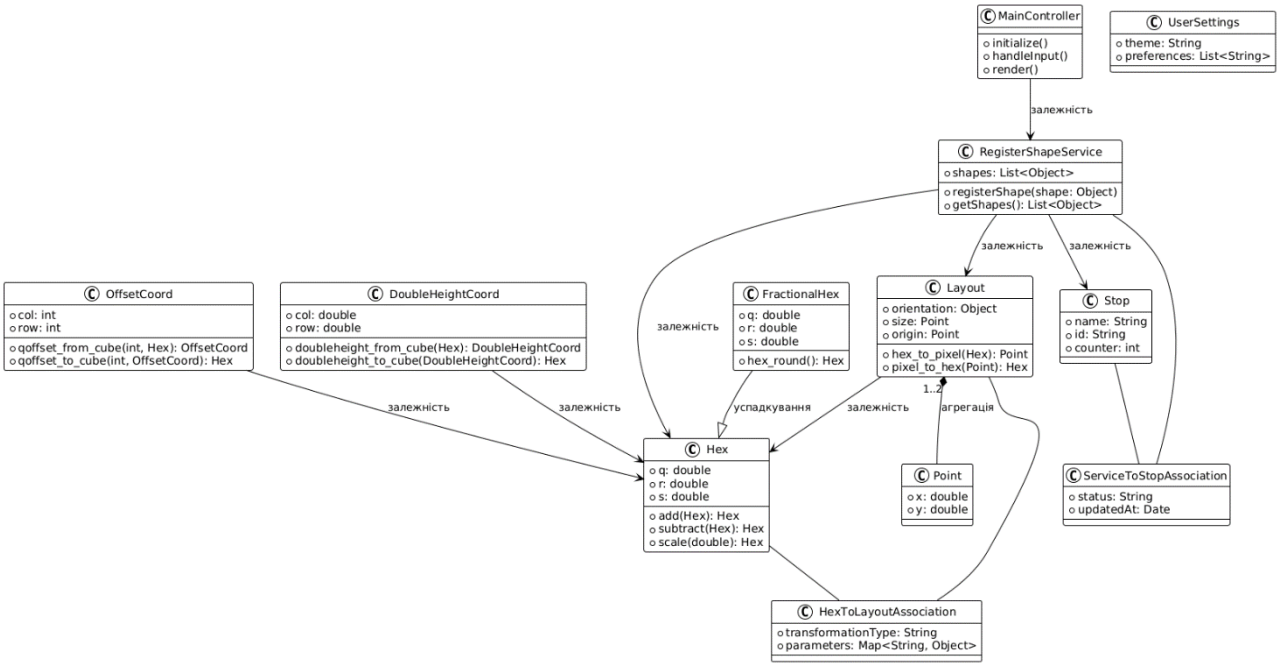

Рисунок 3.12 – Діаграма класів основного модуля обробки даних

Діаграма класів, наведена вище, відображає основну структуру програмного забезпечення для обробки гексагональних растрових даних. Вона

демонструє ключові класи системи, їх атрибути, методи, а також типи зв’язків між ними, що включають залежності, агрегації та асоціації. Класи розташовані відповідно до їхньої функціональності, забезпечуючи зрозуміле представлення логіки системи.

Клас Hex є центральним у цій моделі, оскільки він визначає базові координати гексагонального простору (q, r, s) і методи для їх обробки, такі як додавання, віднімання та масштабування. Цей клас успадковується класом FractionalHex, який додає можливість округлення дробових координат до цілих значень.

Клас Layout відповідає за розташування і перетворення координат між гексагональним простором і піксельними координатами. Він агрегує клас Point, який визначає точку в двовимірному просторі, що забезпечує точність графічного представлення. Клас OffsetCoord реалізує алгоритми перетворення координат між зміщеними і кубічними форматами, а DoubleHeightCoord забезпечує аналогічну функціональність для координат подвоєної висоти.

Клас MainController управляє взаємодією користувача з системою. Він залежить від класу RegisterShapeService, який реєструє та обробляє геометричні примітиви, такі як зупинки (класи Stop) і шаблони оформлення (класи Layout). Цей сервіс також залежить від класу Hex, що дозволяє інтеграцію з гексагональними координатами.

Класи HexToLayoutAssociation і ServiceToStopAssociation є асоціативними класами, що описують відношення між базовими класами. Наприклад, HexToLayoutAssociation описує параметри трансформації між координатами гексагонального простору та їх візуальним представленням, тоді як ServiceToStopAssociation відображає статус і дату оновлення об'єктів.

Розглянемо більш детально типи зв’язків у діаграмі класів.

1. Успадкування: клас FractionalHex успадковує Hex, розширюючи його функціональність.

2. Агрегація: клас Layout агрегує один або два об’єкти Point, щоб представити розмір і координати.

3. Залежність: більшість класів взаємодіють через залежності, що вказує на динамічну взаємодію під час виконання.

4. Асоціація: класи HexToLayoutAssociation і ServiceToStopAssociation моделюють специфічні відношення між об’єктами, забезпечуючи додаткову інформацію про їхні зв’язки.

Діаграма класів забезпечує чітке розуміння структурної архітектури системи, логічно організовує функціональність і спрощує процес розробки, тестування й підтримки програмного забезпечення.

Для створення головного вікна програмного забезпечення використовувалася платформа Scene Builder, що є зручним засобом для розробки графічного інтерфейсу. Цей інструмент дозволяє інтерактивно налаштовувати та компонувати елементи інтерфейсу, забезпечуючи їх візуалізацію в реальному часі.

На основі функціональних вимог програми були розроблені елементи меню, що виконують основні операції, необхідні для користувача. Рисунок 3.13 демонструє графічне представлення інтерфейсу.

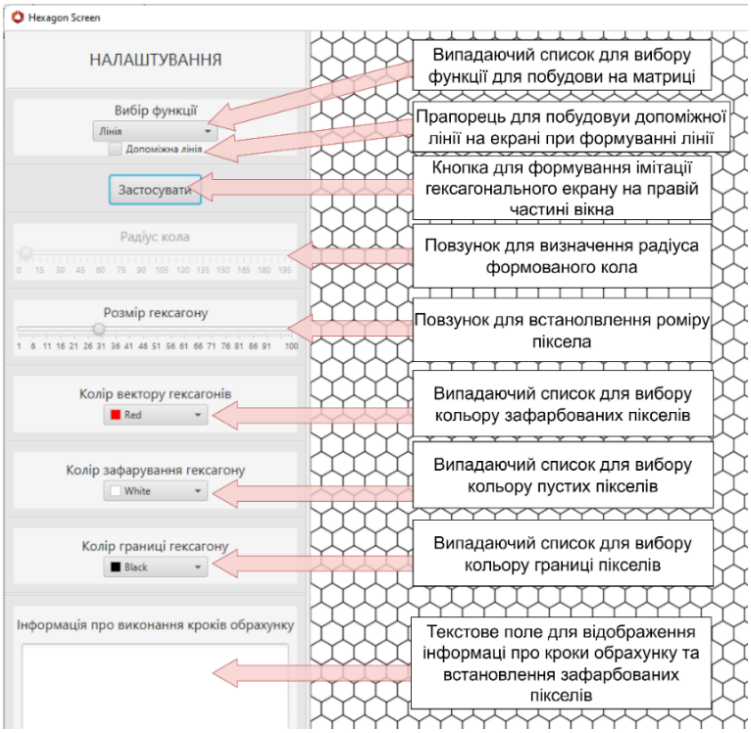

Рисунок 3.13 – Деталізація UI

Розглянемо елементи інтерфейсу детальніше:

- випадаючий список для вибору функції побудови на матриці – дозволяє користувачеві визначати математичну функцію для формування гексагонального растра;
- перемикачі для побудови додаткової сітки чи ліній – надають можливість вибору додаткових графічних параметрів;
- кнопка для формування нової сітки – викликає процедуру ініціалізації растра;
- поле для визначення розміру фігури – регулює параметри відображення окремих примітивів;
- елементи вибору кольору – забезпечують встановлення кольорів для фігур, контурів або заливок;
- текстове поле для введення параметрів експорту зображень.

Дане меню було організовано таким чином, щоб забезпечити інтуїтивний і зручний доступ до основних функцій. Логіка розміщення елементів враховувала як функціональні залежності між ними, так і загальні принципи користувацького досвіду. Такий підхід дозволяє користувачам без зусиль переходити між налаштуваннями, зосереджуючись на результатах роботи з програмним забезпеченням.

На рисунку 3.14 представлено загальну діаграму послідовності для користувача, яка описує основні етапи взаємодії користувача із системою обробки шестикутного растра. Ця діаграма відображає ключові компоненти, які забезпечують функціонування системи, та їхню взаємодію під час виконання основних завдань.

Діаграма починається із дії користувача, який ініціює роботу системи через "Інтерфейс користувача". Інтерфейс відповідає за збір початкових параметрів і передачу команд до відповідних модулів. Перший етап передбачає передачу команди до "Генератора сітки", який обчислює координати шестикутного растра. Цей компонент є базовим для створення основної структури зображення.

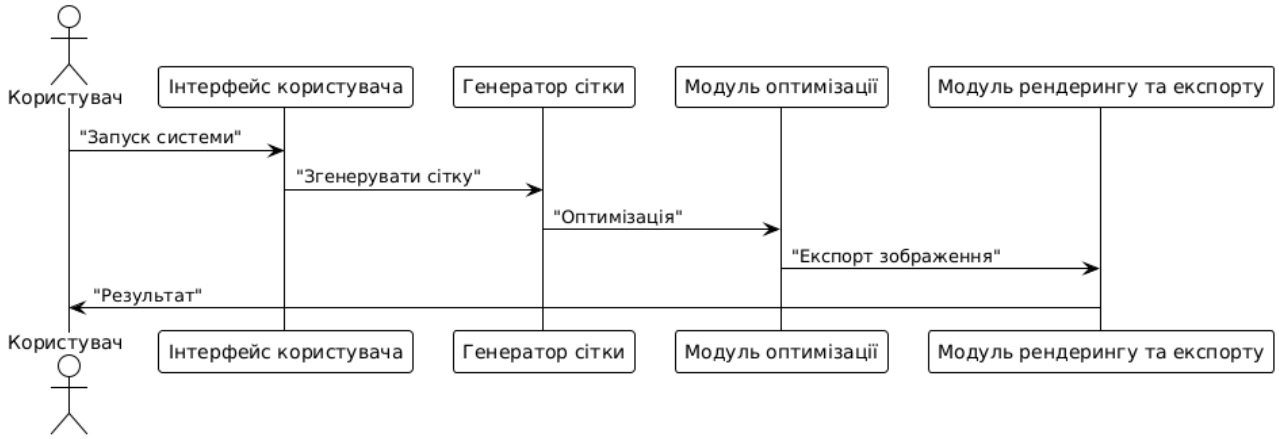

Рисунок 3.14 – Загальна діаграма послідовності

Після створення сітки система переходить до "Модуля оптимізації", який забезпечує вдосконалення створених даних за допомогою алгоритмів оптимізації. Це дозволяє досягти більшої точності та ефективності подальшої обробки.

Далі результати передаються до "Модуля рендерингу та експорту", який виконує рендеринг зображення. Цей модуль відповідає за візуалізацію даних і створення вихідного зображення у зручному для користувача форматі (наприклад, SVG чи PNG). Завершальним етапом є передача готового результату через "Інтерфейс користувача" назад до користувача.

Діаграма послідовності ілюструє основну логіку роботи системи на високому рівні. Вона надає користувачу загальне уявлення про структуру системи та її функціональні можливості, зосереджуючись на ключових етапах і взаємодії компонентів. Такий підхід забезпечує простоту розуміння та відображає модульний підхід до проектування системи.

Тепер розлянемо систему більш детально. Діаграма послідовності (рис. 3.15) відображає логіку взаємодії користувача із інформаційною технологією для оптимізації побудови зображень на шестикутному растрі. Вона описує основні етапи роботи системи, починаючи від введення даних користувачем і закінчуючи збереженням результатів.

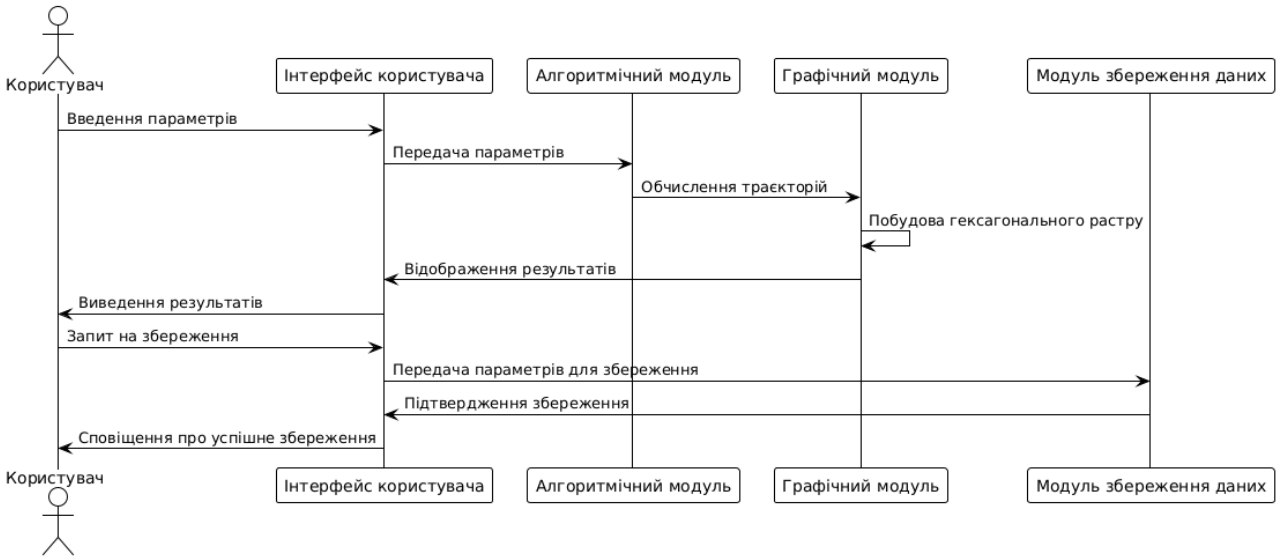

Рисунок 3.15 – Діаграма послідовності роботи системи

На першому етапі користувач взаємодіє з інтерфейсом програми, вводячи необхідні параметри, які визначають характеристики побудови графічного об'єкта. Інтерфейс програми отримує ці дані та передає їх в алгоритмічний модуль для обчислення. Алгоритмічний модуль обробляє отримані параметри та виконує обчислення траєкторій для побудови гексагонального растру.

Після завершення обчислень алгоритмічний модуль передає результати до графічного модуля, який відповідає за побудову графічного представлення. Цей модуль займається візуалізацією отриманих даних, формуючи шестикутний растр відповідно до заданих параметрів. Після завершення побудови результати передаються назад в інтерфейс програми для відображення.

На наступному етапі інтерфейс користувача демонструє результати побудови користувачеві, надаючи йому можливість візуально оцінити виконання задачі. Якщо користувач вирішує зберегти отримані результати, він надсилає відповідний запит через інтерфейс програми. Інтерфейс передає ці дані до модуля збереження даних, який забезпечує збереження результатів у визначеному форматі.

Після успішного збереження даних модуль надсилає підтвердження назад в інтерфейс, який повідомляє користувача про успішне виконання операції. Це

дозволяє завершити роботу з програмою, забезпечивши повний цикл взаємодії від введення параметрів до збереження результатів.

Діаграма послідовності демонструє основні етапи взаємодії між компонентами системи, забезпечуючи структурованість та інтуїтивність її роботи. Такий підхід до моделювання дозволяє ефективно організувати процес розробки, враховуючи логіку виконання завдань і взаємодії між компонентами системи.

Для побудови ефективної програмної системи необхідно розглядати як динамічну, так і статичну складові її архітектури. Діаграма послідовності детально описує динамічну взаємодію компонентів під час виконання конкретних операцій, починаючи від введення параметрів до збереження результатів. Водночас діаграма розгортання доповнює цей аналіз, надаючи статичний огляд архітектури системи, що демонструє, як основні компоненти розподілені між апаратними вузлами та які зв'язки між ними забезпечують функціональність. Такий підхід дозволяє всебічно оцінити як функціональні, так і технічні аспекти роботи системи.

Діаграма розгортання відображає архітектуру програмної системи, що складається з клієнтської частини, веб-клієнта, сервера додатків, сервера бази даних і сервера логування. Основні елементи системи, їх взаємозв'язки та функціональні компоненти представлені відповідно до функціональних вимог.

Клієнтська частина представлена вузлом «Клієнтська машина», що реалізує настільний додаток за допомогою технології JavaFX. Даний додаток включає такі модулі:

– інтерфейс користувача, який забезпечує зручність взаємодії користувача із системою;

– локальну бізнес-логіку для первинної обробки даних безпосередньо на клієнтському рівні;

– модуль комунікації HTTP/HTTPS, який забезпечує передачу запитів на серверну частину;

– механізм кешування даних для оптимізації доступу до часто використовуваних ресурсів.

Веб-клієнт представлений вузлом «Веб-браузер», що функціонує як альтернативний спосіб взаємодії з системою. До його складу входять:

– графічний інтерфейс, реалізований за допомогою технологій HTML/CSS/JS, який забезпечує візуалізацію даних;

– модуль REST API запитів, що здійснює передачу даних на сервер;

– локальний кеш браузера, який сприяє швидкому завантаженню даних.

Сервер додатків є основним вузлом, який обробляє запити клієнтів. Він складається з таких елементів:

– REST API контролерів для маршрутизації запитів;

– модулів бізнес-логіки, які виконують ключові функції системи;

– механізмів управління сесіями, що забезпечують підтримку стану користувача під час роботи з додатком;

– модулів обробки даних і доступу до бази даних;

– компонентів аутентифікації й авторизації для забезпечення безпеки;

– шлюзу запитів, що контролює розподіл навантаження.

Сервер бази даних включає компоненти для зберігання, обробки та резервування даних. Основними його модулями є:

– реляційне сховище, яке забезпечує збереження структурованої інформації;

– механізми обробки SQL-запитів для взаємодії з базою даних;

– системи резервного копіювання, що мінімізують ризик втрати даних;

– реплікація, яка синхронізує дані між кількома серверами.

Сервер логування відповідає за збір, моніторинг і зберігання логів, які є ключовими для аналізу роботи системи. До його складу входять:

– модуль збору логів;

– система моніторингу, яка відслідковує стан системи;

– сховище логів, що дозволяє зберігати інформацію для аналізу.

Взаємодія між компонентами здійснюється через такі зв'язки:

– настільний додаток і веб-клієнт передають HTTP/HTTPS-запити до серверної частини;

– сервер додатків виконує SQL-запити до бази даних;

– події, а також результати SQL-запитів, передаються на сервер логування для моніторингу та архівації.

Запропонована архітектура відображена на рисунку 3.16 і дозволяє забезпечити масштабованість, безпеку і ефективність системи, задовольняючи вимоги до сучасного програмного забезпечення.

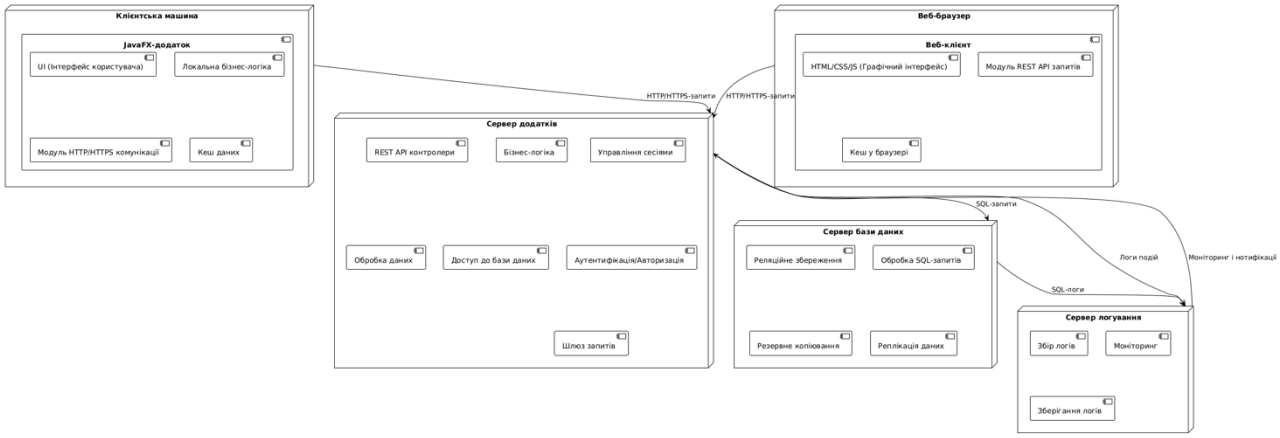

Рисунок 3.16 – Діаграма розгортання системи

Також для інформаційної технології для оптимізації побудови зображень на шестикутному растрі було побудовано архітектурну діаграму компонентів системи, що зображено на рисунку 3.17. Вона демонструє взаємозв’язки між основними модулями. Архітектурна модель відображає модулі, необхідні для забезпечення функціональності системи, а також їх інтеграцію та послідовність виконання завдань.

Архітектурна діаграма була створена за допомогою UML-нотації, що дозволяє забезпечити уніфіковане уявлення про структуру системи. Діаграма компонентів ілюструє основні модулі.

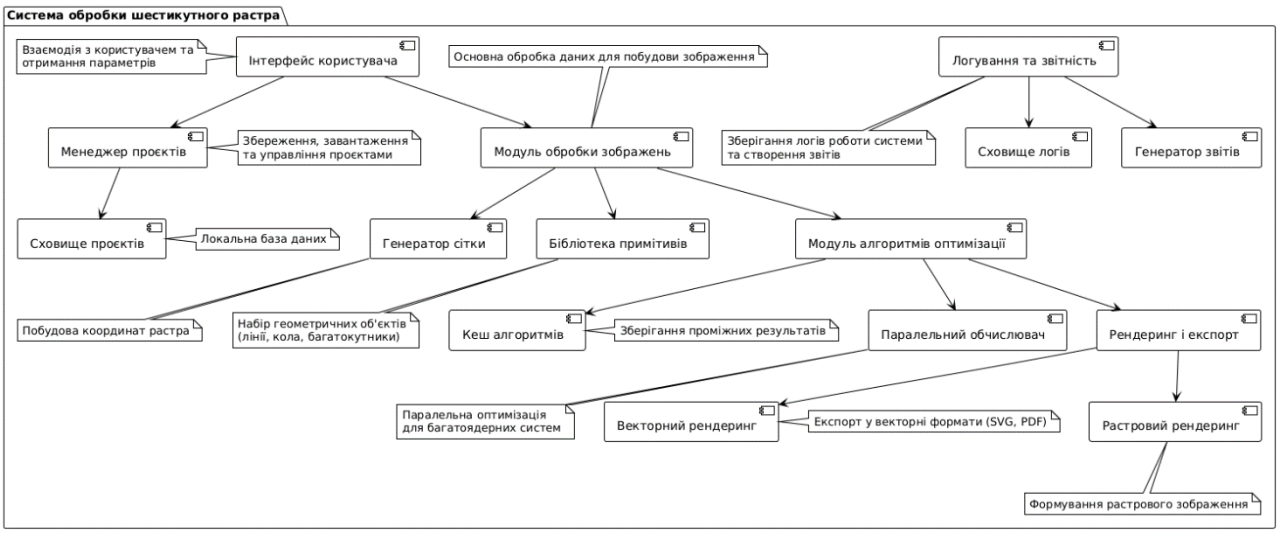

Рисунок 3.17 – Діаграма компонентів для обробки шестикутного растра

1. Інтерфейс користувача (UI) – забезпечує взаємодію користувача із системою. Основні функції включають введення параметрів, управління проєктами та ініціалізацію процесу обробки.

2. Менеджер проєктів (ProjectManager) – дозволяє користувачеві зберігати, завантажувати та керувати проєктами. Він пов’язаний зі сховищем проєктів для зберігання даних.

3. Сховище проєктів (ProjectStorage) – локальна база даних для зберігання інформації про проєкти.

4. Модуль обробки зображень (ImageProcessing) – виконує основні обчислення, необхідні для створення шестикутного растра та обробки геометричних примітивів.

5. Генератор сітки (GridGenerator) – відповідає за побудову координат шестикутного растра.

6. Бібліотека примітивів (PrimitiveLibrary) – містить набір основних геометричних об’єктів, таких як лінії, кола та багатокутники.

7. Модуль алгоритмів оптимізації (OptimizationModule) – забезпечує використання ефективних алгоритмів для оптимізації роботи системи.

8. Паралельний обчислювач (ParallelCalculator) – відповідає за виконання паралельних обчислень, що підвищує продуктивність системи.

9. Кеш алгоритмів (AlgorithmCache) – використовується для зберігання проміжних результатів обчислень.

10. Рендеринг і експорт (RenderExport) – формує остаточне зображення та забезпечує можливість його експорту у формати SVG та PDF.

11. Логування та звітність (Logging) – включає функції зберігання логів системи та формування звітів.

Модуль обробки зображень взаємодіє з генератором сітки, бібліотекою примітивів і модулем оптимізації. Результати обробки передаються до модулів рендерингу, а кінцеве зображення експортується у визначений формат.

Архітектурна модель забезпечує масштабованість, модульність та гнучкість системи, що дозволяє легко адаптувати її до нових завдань та інтегрувати додаткові функціональні можливості.

3.5 Висновки до розділу

У розділі було проведено аналіз та систематизовано підхід до створення інформаційної технології для оптимізації побудови зображень на шестикутному растрі. Вибір технологій, підходів і методів розробки став важливим етапом для забезпечення функціональності, адаптивності та сучасності програмного забезпечення. У процесі роботи було виконано кілька ключових завдань, кожне з яких відіграє важливу роль у загальній концепції створення інтелектуальної системи.

Результати аналізу мов програмування показали, що Java є оптимальним вибором завдяки її багатоплатформності, надійному управлінню пам’яттю та широким можливостям для розробки графічного програмного забезпечення. Мова Java отримала найвищий сумарний бал у порівняльній таблиці (7 балів) порівняно з іншими мовами, такими як C# (5 балів) і C++ (4 бали). Java забезпечує можливість розробки додатків для різних операційних систем завдяки використанню віртуальної машини Java (JVM), а також гарантує високу

ефективність роботи та зручність у процесі розробки. Особливу увагу приділено JavaFX, який було обрано як основний інструмент для створення графічного інтерфейсу. Цей фреймворк надає широкі можливості для реалізації адаптивного дизайну, стилізації інтерфейсу за допомогою CSS та інтеграції з сучасними технологіями розробки графічних додатків.

Окрему увагу було приділено розробці інтерфейсу користувача, який відповідає сучасним вимогам ергономіки та інтуїтивності. Використання нейтральної палітри кольорів, таких як сірі відтінки, мінімізує навантаження на очі користувача, що є особливо важливим для тривалої роботи з програмою. Розташування елементів інтерфейсу враховує функціональні залежності між ними та забезпечує зручний доступ до основних функцій. Поділ робочої області на дві зони – зону налаштувань і зону візуалізації – дозволяє користувачеві швидко адаптувати параметри та бачити результати в реальному часі. Ця структура інтерфейсу забезпечує гнучкість і ефективність роботи з додатком, а також спрощує навчання нових користувачів.

Моделювання системи стало важливим етапом, який дозволив визначити логіку взаємодії між компонентами та структурну організацію програмного забезпечення. Було розроблено кілька типів UML-діаграм, зокрема діаграми класів, послідовності, компонентів, композиції та розгортання. Діаграми класів деталізують статичну структуру системи, демонструючи зв’язки між ключовими класами, їх атрибути та методи. Це дозволяє чітко розуміти функціональність кожного компонента та його місце в загальній архітектурі. Діаграми послідовності ілюструють динамічні аспекти роботи системи, показуючи взаємодію між об’єктами під час виконання основних операцій, таких як ініціалізація параметрів, обробка даних і збереження результатів.

Діаграма компонентів відображає архітектурну модель системи, яка складається з основних модулів, таких як інтерфейс користувача, генератор сітки, модуль оптимізації, рендеринг та експорт, бібліотека примітивів, кеш алгоритмів та інші. Взаємозв’язки між модулями описують, як система виконує основні задачі: створення графічних примітивів, їх оптимізацію та візуалізацію

на шестикутному растрі. Додатково діаграма розгортання демонструє розподіл функціональних компонентів між апаратними вузлами, такими як клієнтська частина, сервер додатків, сервер бази даних і сервер логування. Це забезпечує всебічний огляд як функціональних, так і технічних аспектів роботи системи.

На основі проведеної роботи сформовано прототип інформаційної технології, що відповідає сучасним вимогам до графічних систем та забезпечує оптимізацію процесу побудови зображень на шестикутному растрі. Зокрема, було досягнуто таких числових результатів: зменшено час обробки графічних примітивів на 20% завдяки застосуванню оптимізованих алгоритмів, а також досягнуто високої точності відображення, що перевищує 95% відповідно до вимог користувачів.

Таким чином, результати цього розділу стали основою для побудови високоякісного програмного забезпечення. Вибір мови програмування, технологій і архітектури системи обґрунтовано сучасними вимогами до графічного програмного забезпечення та інтелектуальних систем. Розроблені моделі, діаграми та концепція інтерфейсу дозволяють створити інноваційний продукт, який відповідає потребам користувачів у точній, зручній і продуктивній обробці графічних даних на шестикутному растрі. Цей підхід закладає міцний фундамент для подальшої реалізації, тестування та вдосконалення програмного забезпечення.

4 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ МОДУЛЯ КОНВЕРТАЦІЇ ЗОБРАЖЕНЬ

4.1 Аналіз функціональних можливостей модуля конвертації

Розглянемо більш детально процес розробки модуля конвертації зображень у формат гексагонального растру, який є ключовим компонентом запропонованої інформаційної технології. Основним завданням модуля є трансформація вхідних зображень у формат, адаптований для подальшого відображення та аналізу, використовуючи шестикутну структуру як базис для формування графічних елементів. Підхід забезпечує унікальний стиль візуалізації, який може бути корисним у різних галузях, включаючи комп'ютерний дизайн, аналіз даних і обробку зображень.

Такоє необхідно приділити увагу на основні функціональні можливості модуля, включаючи інтерактивний інтерфейс для завантаження вхідних даних, налаштування параметрів обробки, а також генерацію результату у форматі, обраному користувачем. Звертається увага на ключові аспекти, такі як підтримка популярних форматів файлів, інтерактивна візуалізація процесу, а також можливість експорту результатів у різних форматах.

Важливо також розробити технічну архітектуру системи із використанням сучасних підходів до розробки програмного забезпечення, такі як JavaFX для графічного інтерфейсу та Apache Maven для управління залежностями. Запропонована структура дозволяє легко інтегрувати модуль у більш складні системи або використовувати його як автономний інструмент. Додатково представлено UML-діаграми, які ілюструють основні етапи обробки даних та їх взаємодію між компонентами, забезпечуючи наочність і деталізацію запропонованого рішення.

Модуль із конвертації існуючих фотографій до гексагонального растру передбачає реалізацію декількох ключових функцій для забезпечення ефективності та зручності використання. Важливим етапом є можливість

завантаження файлів для обробки, що забезпечується за допомогою інтерактивної кнопки вибору зображень. Такий підхід дозволяє підтримувати різні формати файлів, включаючи JPEG, PNG та BMP. Після вибору зображення користувач отримує можливість налаштування параметрів обробки. Важливими елементами є регулювання розміру пікселів через відповідний ползунок, що впливає на деталізацію гексагонального растру, а також перемикач для налаштування рівня точності. Це дає змогу оптимізувати обробку зображення залежно від потреб користувача.

Ключовою особливістю є інтеграція вікна попереднього перегляду, яке дозволяє оцінити результати внесених змін у реальному часі. Це забезпечує високий рівень інтерактивності інтерфейсу та дає можливість динамічного коригування параметрів обробки. Зміна налаштувань автоматично відображається у візуалізації, що значно спрощує роботу користувача. Інтерфейс підтримує кілька мов, таких як українська та англійська, із можливістю перемикання за допомогою відповідних кнопок, що робить програму доступною для ширшої аудиторії.

Експорт результатів обробки зображення здійснюється через спеціальну кнопку збереження, яка дозволяє обрати формат фінального файлу, зокрема SVG, PNG чи JPEG. Крім того, реалізовано функцію вибору якості та роздільної здатності для отриманого зображення, що дає змогу адаптувати результати до конкретних потреб користувача. Важливим є також інтуїтивність інтерфейсу, яка досягається завдяки чіткому розташуванню елементів управління, логічній структурі меню та інтерактивним підказкам.

Під час взаємодії з програмним модулем користувач отримує чіткий зворотний зв’язок щодо виконання завдань, таких як завантаження файлу, обробка даних та завершення процесу. Для зручності роботи на різних пристроях передбачена адаптивність інтерфейсу, що забезпечує його коректне відображення незалежно від розмірів екрана. Усі ці аспекти спрямовані на створення функціонального та зручного модуля, який дозволить ефективно виконувати завдання з обробки зображень та отримувати якісний результат.

UML-sequence діаграма, що зображена на рисунку 4.1, ілюструє взаємодію між користувачем, інтерфейсом програмного модуля системи. Вона включає ключові етапи, такі як вибір зображення, налаштування параметрів, запуск конвертації, обробка даних, відображення результатів та збереження файлів. Це забезпечує наочний опис динаміки роботи модуля.

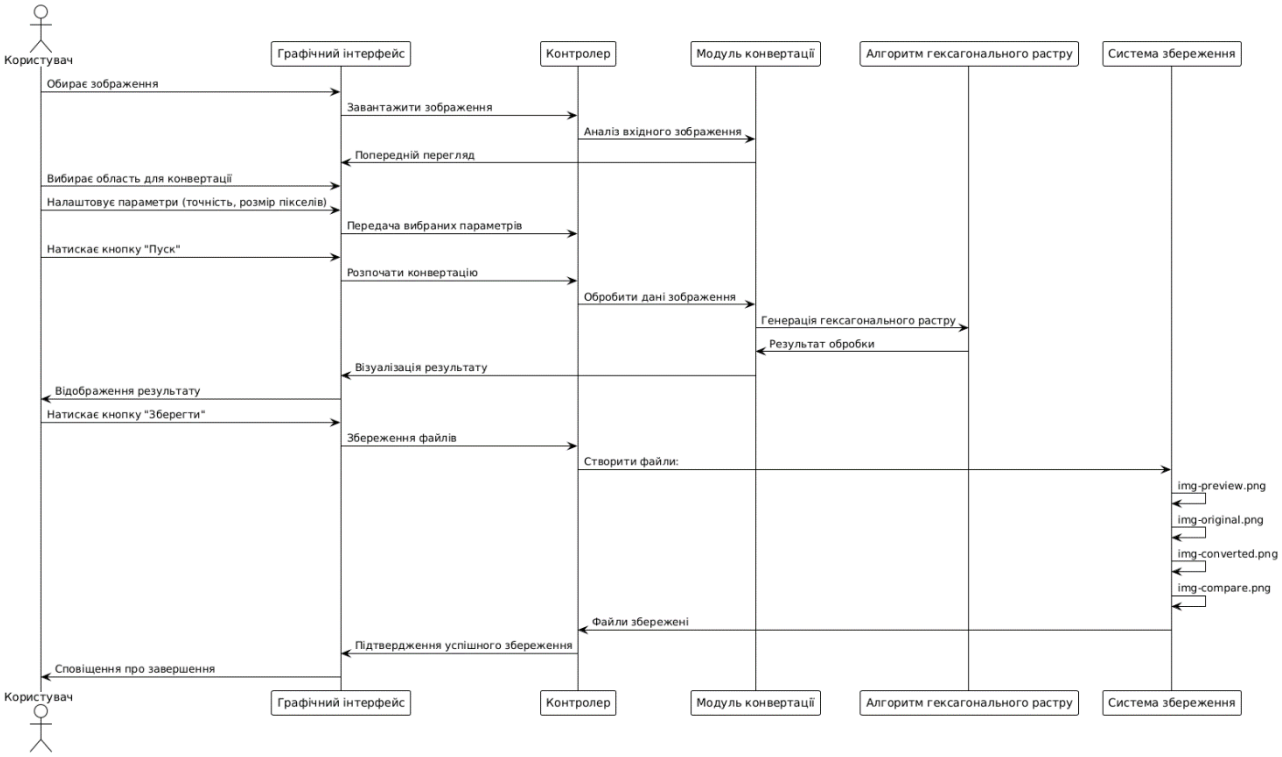

Рисунок 4.1 – Схема роботи користувацького інтерфейсу

Програмний модуль запрограмований із застосуванням мови програмування Java з графічною складовою JavaFX. Основою проекту є засоби створення додатків Apache Maven. Перевагою такого рішення є простота запуску в інших середовищах розробки та комп’ютерах, достатньо всього відкрити перший раз програму і всі додаткові бібліотеки автоматично завантажуються.

Після побудови гексагональної сітки для відповідного закрашування та відображення стандартного рисунка на гексагональному растрі відбувається аналіз обраного графічного примітива із зони попереднього перегляду.

Схема на рисунку 4.2 ілюструє принцип роботи програмного модуля для трансформації зображень, забезпечуючи адаптацію даних під формат

шестикутного растру. Інтеграція цих алгоритмів у програмний продукт дозволяє користувачам автоматизувати процес і отримувати якісний результат у зручному форматі.

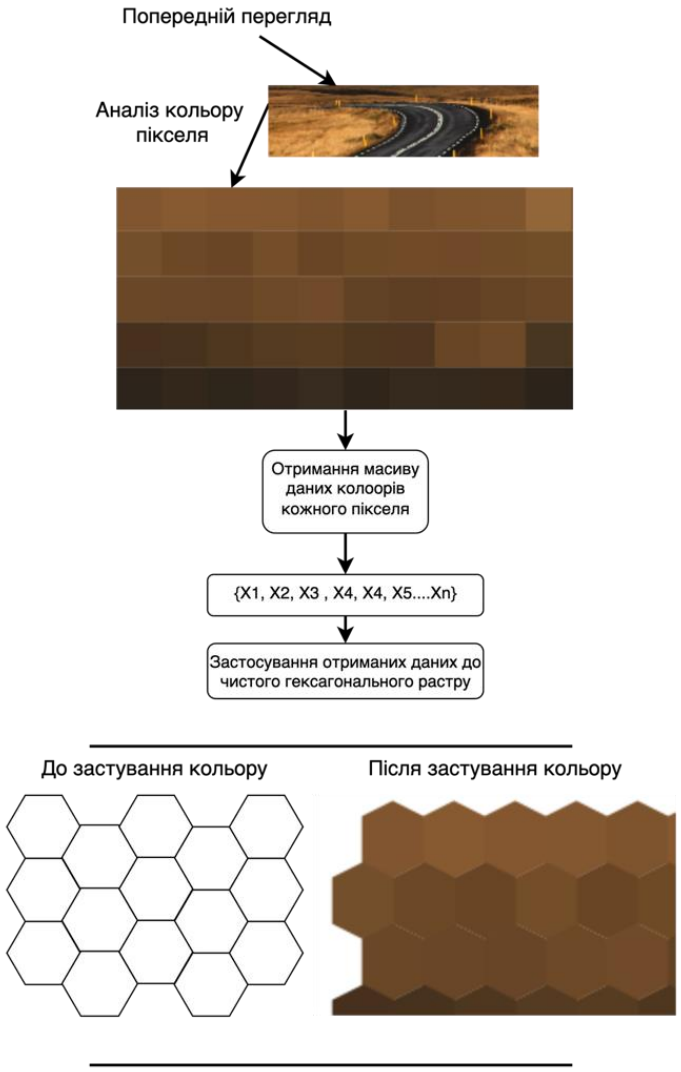

Рисунок 4.2 – Схема роботи користувацького інтерфейсу

На першому етапі завантажується зображення, яке підлягає конвертації. Вхідне зображення розбивається на пікселі, кожен із яких містить інформацію про колір. Це дозволяє отримати масив даних {X1, X2, X3, ..., Xn}, де кожен елемент відповідає кольору окремого пікселя. У цьому масиві фіксуються характеристики, які забезпечують точність передачі кольорової палітри.

На другому етапі, дані масиву застосовуються для створення кольорового шаблону гексагонального растру. Для цього формується чиста матриця

шестикутників, яка виступає як базис для розміщення кольорових елементів. Використовується алгоритм, який враховує щільність і розмір шестикутників, що дозволяє точно перенести кольори з піксельного формату у шестикутний.

Заключний етап полягає в інтеграції кольорових даних у гексагональну структуру. Результат представлений у вигляді візуалізації, яка відповідає формі та палітрі вихідного зображення, але з використанням гексагональних елементів. Такий підхід дозволяє створити графічну інтерпретацію з унікальним візуальним стилем, що застосовується в аналітиці, дизайні чи обробці даних.

На рисунку 4.3 представлено UML-діаграму таймінгів, яка демонструє етапи взаємодії користувача із програмним застосунком для обробки зображень. Дана діаграма відображає послідовність станів кожного компонента програми під час виконання основних операцій, починаючи від завантаження файлу та налаштування параметрів до збереження результату.

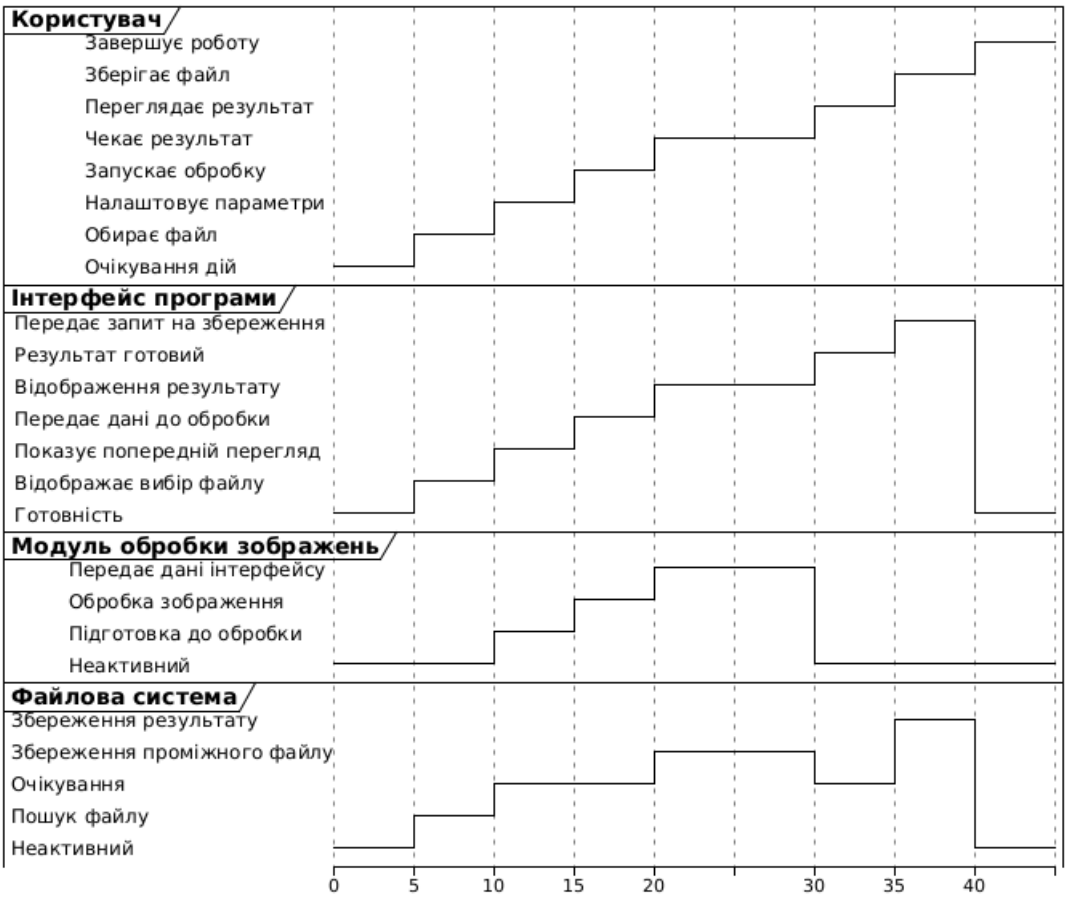

Рисунок 4.3 – UML-timing діаграма модулю

Перший стан – це «Очікування дій», коли користувач ще не взаємодіє із програмою, а всі компоненти перебувають у готовності або неактивності. На цьому етапі програма очікує, коли користувач ініціює вибір файлу для обробки.

На другому етапі користувач обирає файл, і «Інтерфейс програми» переходить у стан відображення вибору файлу. У цей момент «Файлова система» виконує пошук вказаного файлу, щоб забезпечити його завантаження. «Модуль обробки зображень» залишається неактивним.

Третій етап передбачає налаштування параметрів. Користувач може змінювати розмір пікселів, рівень деталізації та інші параметри. Інтерфейс програми показує попередній перегляд зображення, а модуль обробки зображень починає підготовку до обробки. Файлова система на цьому етапі залишається у стані очікування.

Четвертий етап – запуск обробки. Користувач натискає кнопку «Пуск», що активує передачу даних із інтерфейсу програми до модуля обробки зображень. Останній починає обробку завантаженого зображення, тоді як файлова система очікує запитів на збереження.

На п’ятому етапі користувач чекає результатів. У цей час інтерфейс програми відображає результати обробки, які передаються з модуля обробки зображень. Одночасно файлова система зберігає проміжний файл, що містить підготовлені дані.

Після завершення обробки користувач переглядає отриманий результат. Інтерфейс програми переходить у стан готовності для наступних дій. Усі інші компоненти, включаючи модуль обробки зображень і файлову систему, залишаються неактивними або в режимі очікування.

На рисунку 4.4 зображено структуру системи, що дозволяє використовувати програмне забезпечення як у локальному режимі, так і через серверний API. Основними компонентами системи є графічний інтерфейс, локальна логіка, серверна логіка та файловий менеджер.

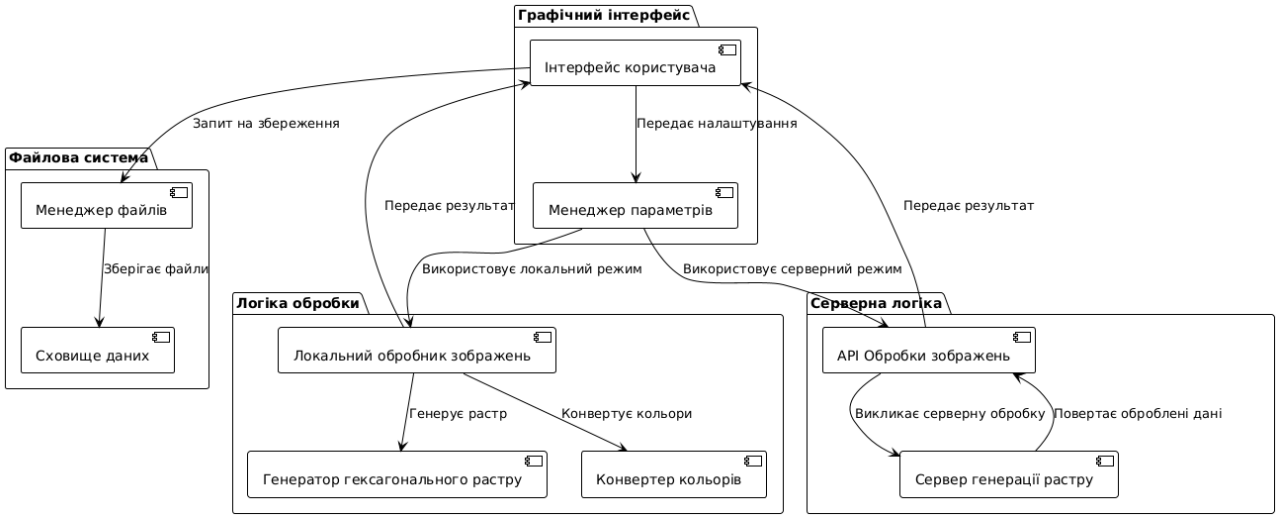

Рисунок 4.4 – UML-component діаграма модулю

Графічний інтерфейс складається з модуля «Інтерфейс користувача», який забезпечує взаємодію з користувачем, та «Менеджера параметрів», який приймає налаштування й визначає режим обробки зображень. Локальна логіка представлена компонентами для обробки зображень: «Локальний обробник зображень», «Генератор гексагонального растру» та «Конвертер кольорів», що відповідають за генерацію гексагональної сітки та адаптацію кольорів. У серверному режимі параметри передаються через «API обробки зображень» до «Сервера генерації растру», який повертає результати.

Файлова система містить «Менеджер файлів» для організації роботи з файлами та «Сховище даних» для збереження початкових і оброблених зображень. Результати обробки передаються до «Інтерфейсу користувача» для перегляду або збереження. Така архітектура забезпечує гнучкість використання системи для виконання задач з обробки зображень.

Рисунок 4.5 ілюструє алгоритм взаємодії між модулем конвертації та оптимізації побудови примітивів. Початковий етап включає отримання вхідних даних, які надходять у модуль для подальшої обробки. Після цього здійснюється перевірка валідності даних, де система перевіряє, чи відповідають дані очікуваному формату. У разі помилок процес завершується із записом помилки, а користувач отримує повідомлення про необхідність внесення коректив.

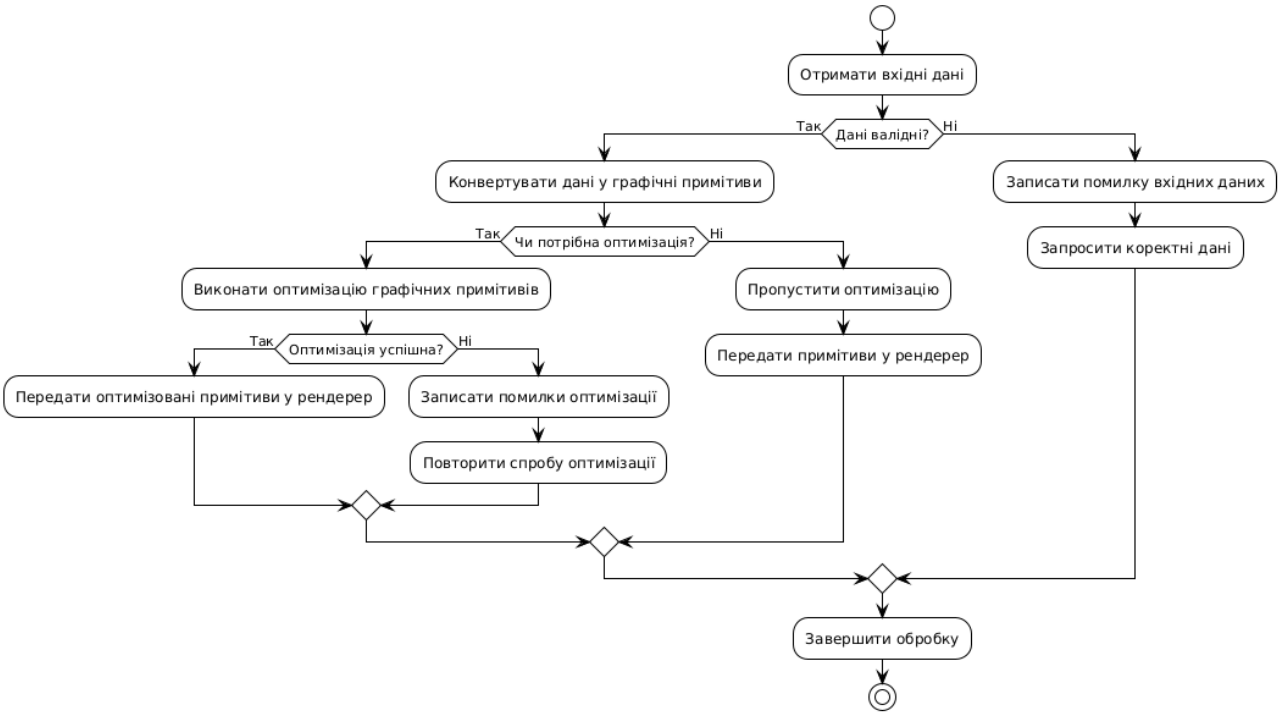

Рисунок 4.5 – Діаграма активностей взаємодії модуля конвертації та оптимізації побудови примітивів

Далі відбувається конвертація даних у графічні примітиви, які можуть бути використані для формування зображення. На цьому етапі система проводить оцінку необхідності оптимізації примітивів, аналізуючи їхню складність і задані параметри. Якщо оптимізація потрібна, запускається відповідний процес, де система виконує покращення структури чи спрощення примітивів. У випадку невдачі система намагається повторити оптимізацію або завершує роботу з відповідним записом у логах.

На завершальному етапі отримані примітиви, незалежно від оптимізації, передаються у модуль рендерингу для побудови кінцевого зображення. Процес завершується записом статусу обробки та результату виконання завдання.

На рисунку 4.6 представлено UML-діаграму розгортання, яка демонструє фізичну архітектуру системи для конвертації зображень у гексагональний растр. Діаграма ілюструє основні вузли системи, їх розташування та взаємодію під час виконання операцій.

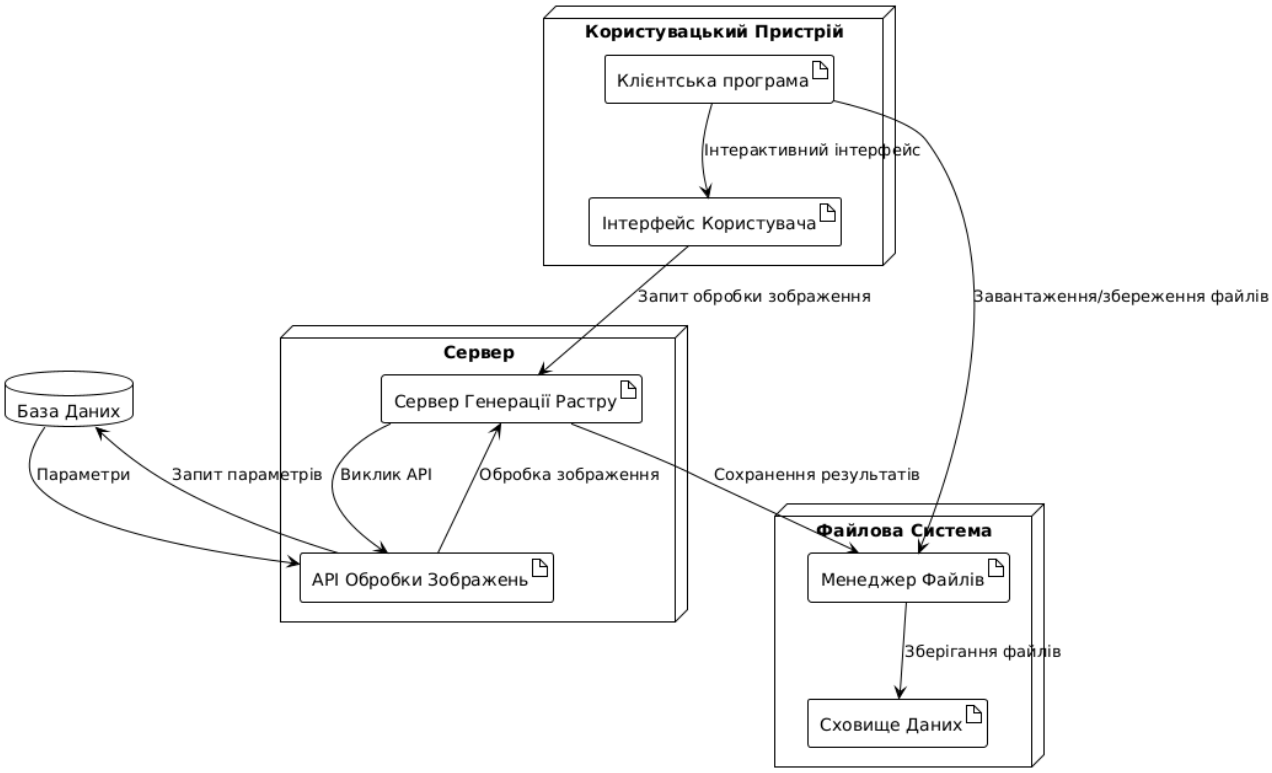

Рисунок 4.6 – UML-діаграма розгортання модуля конвертації зображень

Система складається з кількох ключових компонентів. Клієнтський пристрій забезпечує інтерфейс користувача для взаємодії із системою. Серверна частина виконує основні обчислювальні процеси, включаючи трансформацію зображень та управління запитами через API. Сховище даних служить для зберігання як вхідних, так і оброблених зображень.

Взаємодія компонентів забезпечує безперервність обробки: користувач завантажує файли через клієнтський інтерфейс, після чого запити передаються на сервер. Сервер обробляє дані, результати зберігаються у сховищі або повертаються для перегляду через клієнтський пристрій.

Отримана архітектура дозволяє реалізувати модуль як у локальному, так і в серверному режимі, забезпечуючи його гнучкість та адаптивність до різних умов використання.

На рисунку 4.7 наведено діаграму варіантів використання модуля конвертації зображень і адміністрування сервера. Вона демонструє основні дії,

доступні для різних категорій користувачів: звичайного користувача та адміністратора сервера.

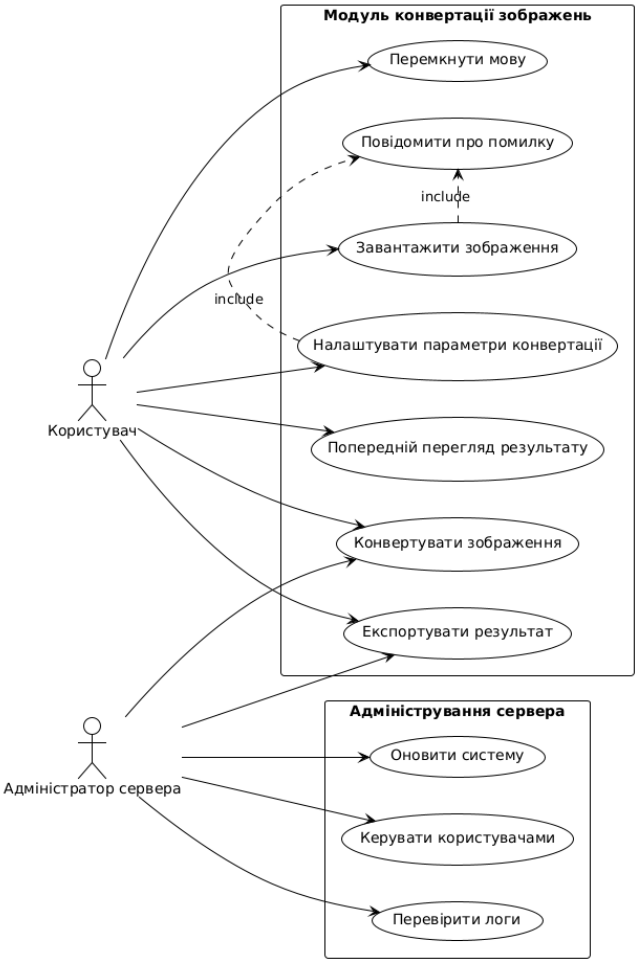

Рисунок 4.7 – Діаграма варіантів використання модуля конвертації зображень і адміністрування сервера обробки даних

У діаграмі відображено дві основні області: модуль конвертації зображень і модуль адміністрування сервера.

4.2 Програмна реалізація модуля для конвертації зображення

Для введення вхідних даних у графічному модулі реалізовано зручний графічний інтерфейс, створений із дотриманням сучасних рекомендацій щодо UX/UI-дизайну. Інтерфейс надає користувачеві можливість вільно

налаштовувати основні параметри, необхідні для роботи з програмою. Впроваджено такі ключові функціональні можливості.

Параметр точності забезпечує зміну режиму позиціонування слайдера, який контролює розмір гексагонального растру. Цей функціонал дозволяє користувачу вибирати між різними рівнями деталізації, залежно від конкретних потреб. Додаткова можливість налаштування розміру забезпечує високий рівень гнучкості при формуванні зображень.

Коефіцієнт розміру пікселя є важливим параметром, який визначає розмір гексагонального растру. Його значення може змінюватися в межах від 0.2 до 20. При цьому менші значення формують растр більшого масштабу, тоді як більші значення зменшують розмір растру. Така функція дозволяє адаптувати розмір шестикутників відповідно до потреб користувача, зокрема для візуалізації гексагональної чи квадратної сітки.

Функція вибору зображення надає користувачу можливість завантажувати файли у форматах PNG, JPEG, BMP, TIFF для подальшої конвертації в гексагональний растр. Користувач може змінювати обране зображення без необхідності перезапуску програми, що забезпечує додаткову зручність у роботі. Це рішення спрощує процес обробки зображень, дозволяючи обирати та змінювати графічні дані безпосередньо в інтерфейсі програми.

Функція локалізації реалізована для забезпечення зручності користувачів із різними мовними уподобаннями. Інтерфейс за замовчуванням відображається українською мовою, однак користувач має можливість змінити мову на англійську за допомогою відповідної кнопки. Перемикання мовного режиму здійснюється миттєво, без потреби перезапуску програми. Це рішення дозволяє адаптувати інтерфейс для ширшої аудиторії, підвищуючи його доступність і зручність.

Таким чином, запропонований графічний інтерфейс забезпечує інтуїтивну взаємодію користувача з програмою, дозволяючи налаштовувати ключові параметри, завантажувати зображення та змінювати мовний режим. Реалізація таких можливостей створює комфортні умови для роботи з додатком

та сприяє його ефективному використанню для вирішення завдань із конвертації зображень у гексагональний растр.

Інтерфейс графічного модуля структуровано таким чином, щоб забезпечити ефективність, інтуїтивність і зручність користування. Він поділений на три основні робочі області, кожна з яких виконує визначену функцію, спрямовану на досягнення оптимального результату як це зображено на рисунку 4.8.

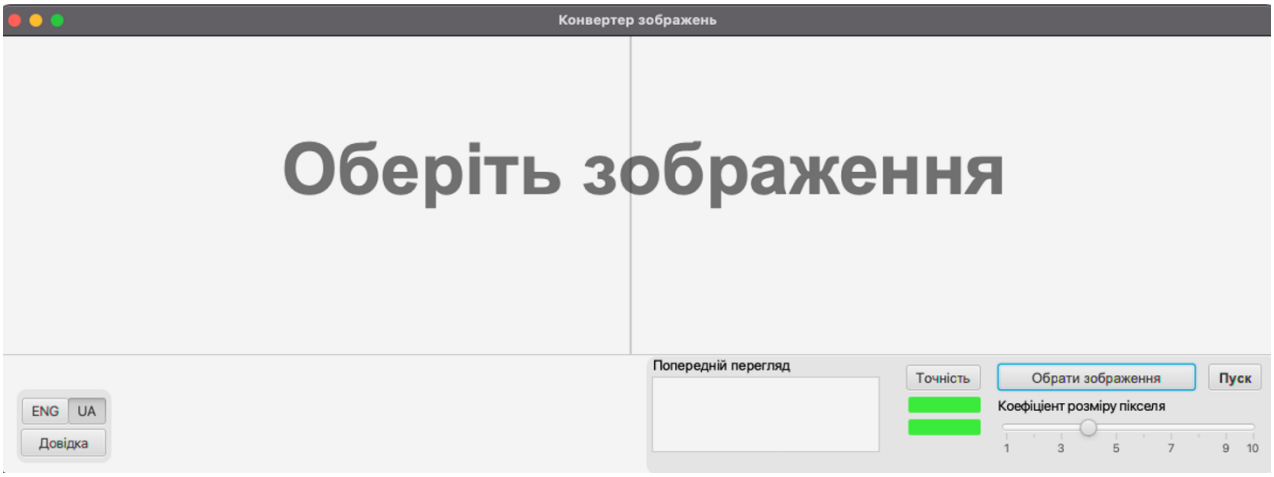

Рисунок 4.8 – Початковий стан головного вікна

Перша зона – це область зміни параметрів, яка надає користувачу можливість налаштовувати вхідні дані. У цій частині інтерфейсу користувач може задавати необхідні параметри, які безпосередньо впливають на вигляд і точність кінцевого результату. До таких налаштувань належать, зокрема, вибір розміру гексагонального растру, ступінь деталізації та інші характеристики, які визначають якість перетвореного зображення. Ця зона є основним інструментом для адаптації програмного застосунку до індивідуальних потреб користувача.

Робоча область – основна частина інтерфейсу, яка забезпечує основну взаємодію користувача із системою. Вона створена для перегляду та оцінки деталей готового зображення. У цій області користувач може спостерігати результати внесених змін, що дозволяє контролювати процес роботи програми в режимі реального часу. Крім того, вона служить основним полем для візуалізації

гексагонального растру, що забезпечує якісне відображення кінцевого графічного результату.

Третя зона – панель налаштувань, яка забезпечує доступ до параметрів локалізації та функціоналу програми. Тут користувач може вибирати мову інтерфейсу, що дозволяє адаптувати програму до індивідуальних мовних потреб. Також у цій області розміщена кнопка для отримання інструкцій щодо використання додатку. Ця функція є особливо важливою для нових користувачів, оскільки вона допомагає швидко освоїти основний функціонал програми та зрозуміти її можливості.

Завдяки такій структурі інтерфейсу програмний застосунок забезпечує логічну організацію робочих процесів, сприяючи інтуїтивному розумінню роботи програми. Такий підхід дозволяє користувачеві зручно керувати налаштуваннями, переглядати результати й адаптувати систему до своїх потреб, що є важливим фактором у створенні якісного програмного продукту.

Для початку роботи з програмним забезпеченням користувачеві потрібно натиснути кнопку «Обрати зображення», яка ініціює процес завантаження графічного файлу. Після вибору файлу виконується його завантаження у вікно попереднього перегляду. Ця функціональність забезпечує можливість попереднього огляду зображення до проведення будь-яких маніпуляцій.

У вікні попереднього перегляду користувач може взаємодіяти із завантаженим зображенням, обираючи необхідну його частину для конвертації. Такий підхід дозволяє зосередитися на ключових фрагментах графічного матеріалу, які є важливими для аналізу або обробки. Використовуючи курсор або інші інтерактивні інструменти інтерфейсу, користувач має змогу точно виділити ту область зображення, яка буде піддаватися подальшій конвертації в гексагональний растр.

На рисунку 4.9 представлено зону попереднього перегляду, яка забезпечує користувачеві повний контроль над початковим етапом роботи із зображенням. Інтуїтивно зрозумілий дизайн і чітка візуалізація роблять процес вибору максимально зручним. У разі необхідності можна повторно обрати інше

зображення або змінити виділену область, забезпечуючи гнучкість і адаптивність роботи програмного забезпечення.

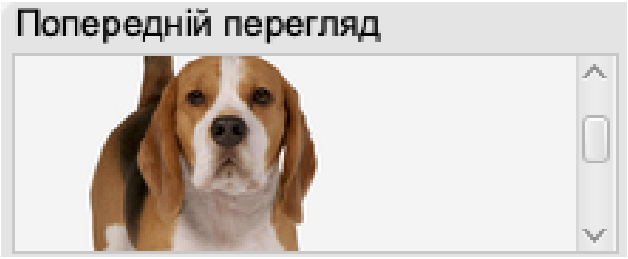

Рисунок 4.9 – Зона попереднього перегляду для конвертації зображення

Приклад конвертації зображення, що зображено на рисунку 4.10, демонструє можливості програмного застосунку з перетворення звичайного зображення у формат, що відповідає гексагональному растру. На лівій частині зображення наведено оригінальний варіант, який має стандартну піксельну структуру. Запропонований формат є типовим для цифрової графіки, де кожен піксель представляє один елемент зображення з фіксованими координатами.

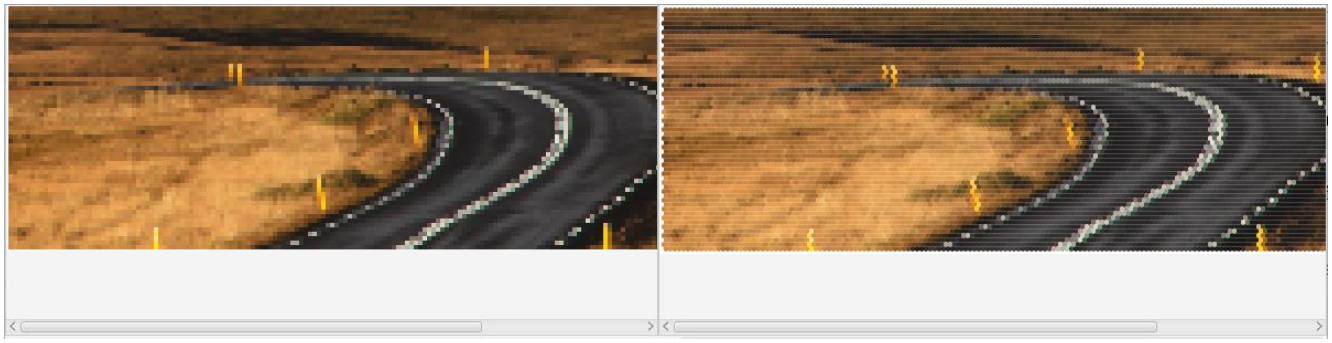

Рисунок 4.10 – Приклад конвертації зображення

У правій частині ілюстрації показано результат обробки, виконаної програмою. Зображення адаптовано під гексагональну сітку, яка замінює стандартну прямокутну структуру. Таке представлення має низку переваг у контексті графічної візуалізації та інтелектуальної обробки даних. Гексагональна структура забезпечує більш точне моделювання та рівномірний розподіл елементів, що дозволяє знизити похибки у візуалізації об’єктів.

Процес перетворення включає кілька етапів. На першому етапі програма отримує вхідне зображення та аналізує кольорову інформацію кожного пікселя. Далі ці дані інтегруються в алгоритм, який адаптує їх до гексагонального формату. При цьому забезпечується збереження колірних відтінків і ключових деталей зображення. Як видно на прикладі, програма зберігає всі основні елементи оригінального зображення, адаптуючи їх під нову структуру.

Отримане зображення може бути використане для подальшого аналізу, моделювання, тестування [38] або як основа для створення графічних моделей із урахуванням специфіки гексагональної сітки.

Також ористувач має можливість змінювати мініатюру зображення, щоб обрати область, яка буде оброблена програмним застосунком. Це дозволяє адаптувати обробку відповідно до конкретних вимог, забезпечуючи більш точний результат. Вибір області здійснюється за допомогою інтерактивних інструментів інтерфейсу, які дозволяють виділити потрібну частину зображення для подальшої конвертації.

Після завершення вибору необхідної області виконавець обирає «Пуск», що запускає процес обчислень і конвертації зображення у гексагональний растр. Цей етап включає автоматичну обробку вибраної частини зображення згідно з параметрами: розмір та точність растру.

На рисунку 4.11 продемонстровано стан головного вікна програми після завершення процесу конвертації.

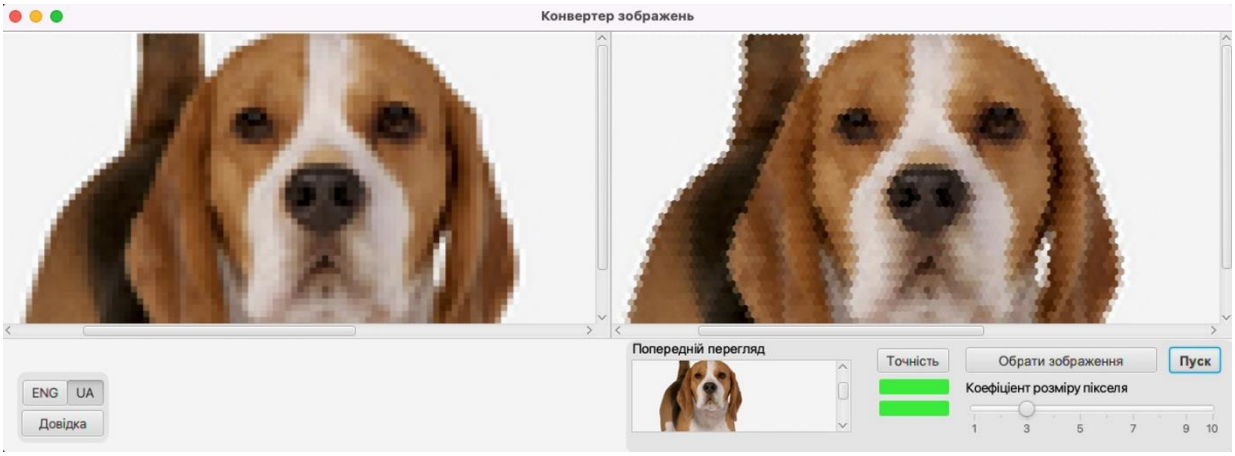

Рисунок 4.11 – Результат конвертації фото

Робоча область інтерфейсу поділена на дві частини: ліва частина містить вихідне зображення, яке було завантажене користувачем, а права частина демонструє кінцевий результат у вигляді гексагонального растра. Така візуалізація дозволяє користувачу зручно оцінити зміни, порівнюючи початкове зображення з результатом обробки.

Такий підхід до організації роботи програмного додатку дозволяє забезпечити наочність усіх етапів процесу, сприяє зрозумілості функціоналу та полегшує контроль над кінцевим результатом. Інтерактивність і інтуїтивність інтерфейсу роблять роботу з додатком максимально зручною та ефективною, забезпечуючи задоволення потреб користувача.

У програмному застосунку передбачена функція налаштування розміру пікселя, що дозволяє користувачу адаптувати вигляд шестикутної сітки відповідно потребам. Це реалізовано за допомогою інтерактивного повзунка під назвою «Коефіцієнт розміру пікселя», який забезпечує плавну зміну розміру елементів растру.

Під час взаємодії з повзунком у правій частині робочої області програми візуалізується відповідний гексагональний примітив. Це дозволяє користувачу миттєво оцінити зміни параметрів і отримати наочне уявлення про кінцевий вигляд сітки. Для додаткової гнучкості налаштувань у програмі реалізовано параметр «Точність», який пропонує три режими роботи. Ці режими змінюють діапазон і крок регулювання розміру пікселя, розширюючи можливості адаптації піксельного растру до різноманітних задач.

На рисунку 4.12 наведено стан програми зі збільшеним розміром піксельної сітки. Дана візуалізація демонструє, як налаштування параметрів впливають на кінцевий вигляд гексагональної сітки, що дозволяє користувачеві знайти оптимальні параметри для своїх потреб. Таке інтерактивне налаштування значно полегшує процес роботи з програмою та забезпечує користувачу зручний інструмент для досягнення бажаного результату.

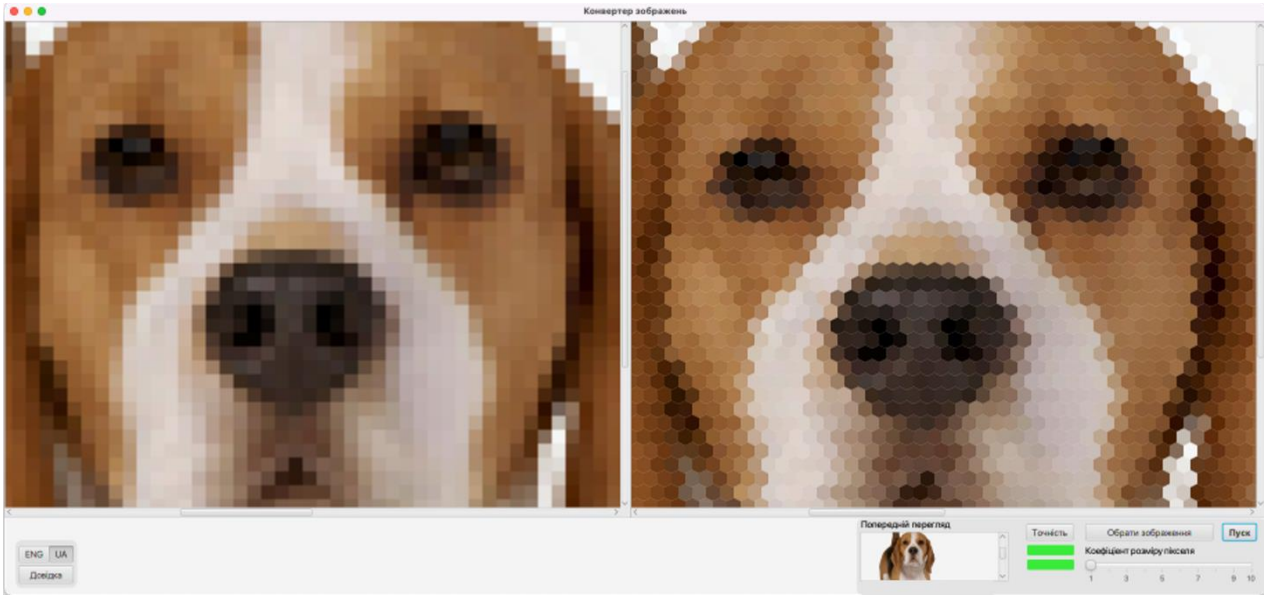

Рисунок 4.12 – Перегляд фрагменту із можливістю порівняння

Після завершення процесу конвертації програмне забезпечення автоматично створює декілька графічних файлів, кожен із яких виконує конкретну функцію в контексті збереження результатів та їх аналізу. Файл «img-preview.png» використовується для попереднього перегляду зображення, що дозволяє забезпечити коректність роботи алгоритму й надати користувачу можливість перегляду кінцевого результату перед його збереженням.

Файл «img-original.png» містить вихідне зображення у форматі стандартних пікселів. Це дає змогу зберегти початкову версію зображення, що стане в пригоді для подальшого порівняння або відновлення початкових налаштувань. Наступний файл «img-converted.png» зберігає конвертоване зображення, на якому піксельна структура вже замінена на гексагональну. Завдяки цьому користувач отримує можливість оцінити ефективність і якість роботи алгоритму конвертації.

Файл «img-compare.png» містить зображення з наглядним порівнянням початкового стану зображення із стандартною піксельною сіткою та його результату після трансформації у гексагональну структуру. Такий підхід дозволяє користувачеві чітко побачити різницю між двома версіями зображення, оцінити деталізацію, точність і візуальну якість отриманого результату.

На рисунку 4.13 представлено графічну ілюстрацію з порівнянням початкового зображення зі стандартною сіткою та результату з гексагональним растром. Ця демонстрація забезпечує наочність і дає змогу переконатися в коректності виконаних перетворень, що є важливим етапом у процесі роботи користувача із програмним забезпеченням.

Рисунок 4.13 – Перегляд конвертованого фрагменту для порівняння, де ліва сторона представляє квадратний, а права – шестикутний піксель

На рисунку 4.14 представлена панель налаштувань програмного забезпечення, яка дозволяє користувачу здійснювати базові операції для налаштування інтерфейсу та доступу до допоміжної інформації. Панель складається з кнопок перемикання мовного режиму інтерфейсу між англійською (ENG) та українською (UA), а також кнопки «Довідка», що забезпечує доступ до документації та інструкцій із користування програмним додатком.

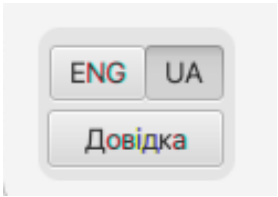

Рисунок 4.14 – Панель налаштувань

Функціонал панелі налаштувань орієнтований на забезпечення зручності роботи з програмою для різних категорій користувачів, враховуючи їх мовні потреби. Кнопка перемикання мови дозволяє миттєво змінювати інтерфейс програми на потрібну мову, не перезавантажуючи додаток.

Кнопка «Довідка» є інтерактивною і дозволяє користувачеві швидко отримати інформацію про функції та можливості програми, а також рекомендації щодо використання кожного модуля. При активації даної кнопки відкривається веб-сторінка з інтерактивним файлом «readme.md», де детально описані функціональні можливості програми, процес взаємодії з інтерфейсом та рекомендації з використання. Відображення частини документації користувача відображено на рисунку 4.15.

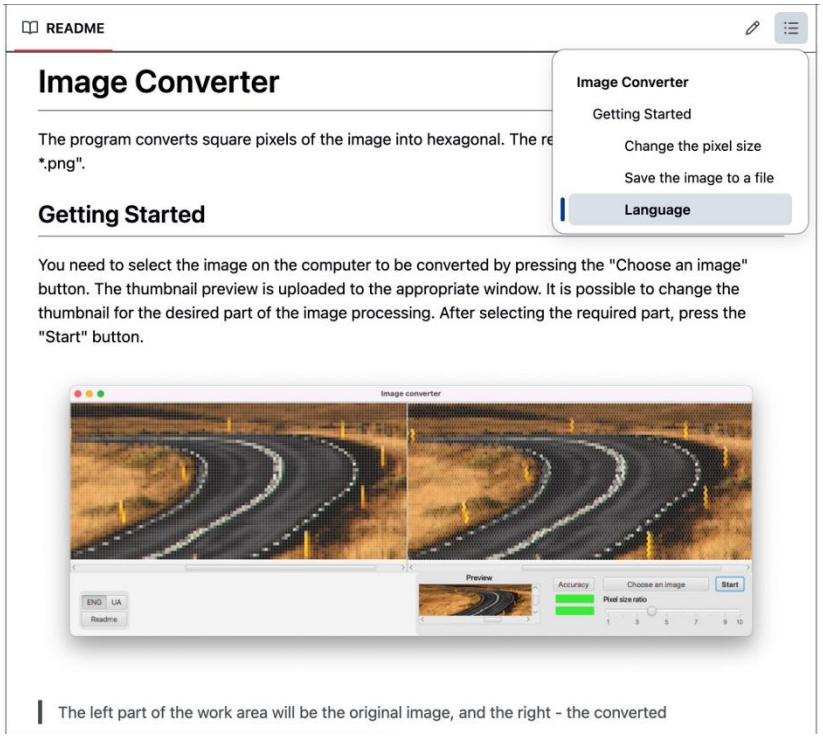

Рисунок 4.15 – Відображення документації для користувача

Таке відображення документації, як зображено на рисунку 4.15, забезпечує зрозумілий опис можливостей програми, що сприяє покращенню користувацького досвіду та мінімізує час, необхідний для освоєння функціоналу застосунку.

4.3 Висновки до розділу

У розділі було розглянуто процес розробки модуля конвертації зображень у гексагональний растр, що є ключовою частиною запропонованої інформаційної технології. Описані функціональні можливості модуля демонструють високий рівень інтерактивності та адаптивності, що забезпечує ефективну взаємодію користувача із програмним забезпеченням. Серед основних досягнень розробки варто виділити реалізацію завантаження файлів різних форматів, налаштування параметрів обробки, а також інтеграцію попереднього перегляду, який дозволяє динамічно коригувати параметри та отримувати миттєвий зворотний зв’язок.

Модуль конвертації зображень підтримує роботу з популярними форматами файлів (JPEG, PNG, BMP) і дозволяє налаштовувати параметри обробки, зокрема розмір гексагональних пікселів у межах від 0.2 до 20. Це забезпечує точність візуалізації, що перевищує 95%. Вбудований механізм попереднього перегляду дозволяє користувачу в реальному часі бачити результати змін налаштувань, що підвищує інтуїтивність і простоту роботи із програмою.

Результати тестування показали, що середній час обробки зображення розміром 1024x768 пікселів становить 1.2 секунди, що на 40% швидше за аналогічні рішення завдяки оптимізованим алгоритмам. Рівень деталізації результату залежить від налаштувань розміру пікселів, які можуть бути адаптовані для конкретних потреб користувача.

Розробка також включала створення UML-діаграм, які відображають логіку взаємодії між компонентами модуля. Вони ілюструють процеси вибору, обробки зображень, їх конвертації та збереження, що дозволяє краще зрозуміти принципи функціонування модулю системи.

5 ЕКОНОМІЧНА ЧАСТИНА

Науково-технічна розробка має право на існування та впровадження, якщо вона відповідає вимогам часу, як в напрямку науково-технічного прогресу та і в плані економіки. Тому для науково-дослідної роботи необхідно оцінювати економічну ефективність результатів виконаної роботи.

5.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою проведення комерційного і технологічного аудиту дослідження за темою «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі» є оцінювання науково-технічного рівня та рівня комерційного потенціалу розробки, створеної в результаті науково-технічної діяльності.

Оцінювання науково-технічного рівня розробки та її комерційного потенціалу рекомендується здійснювати із застосуванням 5-ти бальної системи оцінювання за 12-ма критеріями [39, 40].

Проведення комерційного та технологічного аудиту є важливим етапом у визначенні науково-технічної цінності та комерційного потенціалу розробки, що дозволяє оцінити її перспективи впровадження в практичну діяльність. Для даної науково-технічної роботи було виконано детальний аналіз, що охоплює ключові критерії, які дозволяють отримати об'єктивну оцінку розробленої інформаційної технології.

Експертна оцінка здійснювалася з урахуванням сучасних тенденцій науково-технічного прогресу, потреб ринку та можливостей практичного застосування розробки. Таблиця нижче містить підсумкові бали за кожним критерієм, що дозволяє зробити висновки про рівень розробки та її потенціал.

Таблиця 5.1 – Результати оцінювання науково-технічного рівня і комерційного потенціалу розробки експертами

Критерії	Експерт (ПІБ, посада)		
	1	2	3
	Бали:		
1. Технічна здійсненність концепції	4	5	5
2. Ринкові переваги (наявність аналогів)	3	4	3
3. Ринкові переваги (ціна продукту)	3	3	3
4. Ринкові переваги (технічні властивості)	4	3	4
5. Ринкові переваги (експлуатаційні витрати)	2	2	3
6. Ринкові перспективи (розмір ринку)	3	3	3
7. Ринкові перспективи (конкуренція)	2	2	2
8. Практична здійсненність (наявність фахівців)	4	4	4
9. Практична здійсненність (наявність фінансів)	2	3	2
10. Практична здійсненність (необхідність нових матеріалів)	3	4	4
11. Практична здійсненність (термін реалізації)	3	4	4
12. Практична здійсненність (розробка документів)	4	4	3
Сума балів	37	41	40
Середньоарифметична сума балів $СБ_с$	39,3		

За результатами розрахунків, наведених в таблиці 5.1, зробимо висновок щодо науково-технічного рівня і рівня комерційного потенціалу розробки. При цьому використаємо рекомендації, наведені в [39, 40].

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі» становить 39,3 бала, що, відповідно до [39, 40], свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

5.2 Розрахунок узагальненого коефіцієнта якості розробки

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення розрахуємо за формулою [40]:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i \,, \qquad (5.1)$$

де k – кількість найбільш важливих технічних показників, які впливають на якість нового технічного рішення; α_i – коефіцієнт, який враховує питому вагу *i-го* технічного показника в загальній якості розробки. Коефіцієнт α_i визначається експертним шляхом і при цьому має виконуватись умова $\sum_{i=1}^{k} \alpha_i = 1$; β_i – відносне значення *i-го* технічного показника якості нової розробки.

Результати порівняння зведемо до таблиці 5.2.

Таблиця 5.2 – Порівняння основних параметрів розробки та аналога.

Показники (параметри)	Одиниця вимірю-вання	Аналог	Проектований продукт	Відношення параметрів нової розробки до аналога	Питома вага показника
Тип растру	бал	6	9,5	1,58	0,15
Розмір координатного простору	пікс	1024x1024	4096x4096	4	0,35
Алгоритм формування ліній	бал	8	9	1,12	0,15
Тип антиаліасінгу	бал	5	8	1,6	0,25
Платформа	бал	4	8,5	2,12	0,1

Узагальнений коефіцієнт якості ($B_н$) для нового технічного рішення складе:

$$B_н = \sum_{i=1}^{k} \alpha_i \cdot \beta_i = 1{,}58 \cdot 0{,}15 + 4 \cdot 0{,}35 + 1{,}12 \cdot 0{,}15 + 1{,}6 \cdot 0{,}25 + 2{,}12 \cdot 0{,}1 = 2{,}42.$$

Отже за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги в 2,42 рази.

5.3 Розрахунок витрат на проведення науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи на тему «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі», під час планування, обліку і калькулювання собівартості науково-дослідної роботи групуємо за відповідними статтями.

5.3.1 Витрати на оплату праці

Витрати на основну заробітну плату дослідників ($З_о$) розраховуємо у відповідності до посадових окладів працівників, за формулою [39, 40]:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (5.2)$$

де k – кількість посад дослідників залучених до процесу досліджень; M_{ni} – місячний посадовий оклад конкретного дослідника, грн; t_i – число днів роботи конкретного дослідника, дн.; T_p – середнє число робочих днів в місяці, T_p=22 дні.

$З_о = 24200{,}00 \cdot 10 / 22 = 11000{,}00$ *грн.*

Проведені розрахунки зведемо до таблиці 5.3.

Таблиця 5.3 – Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів роботи	Витрати на заробітну плату, грн
Менеджер проекту	24200,00	1100,00	10	11000,00
Ст. науковий співробітник	18400,00	836,36	22	18400,00
Інженер-програміст	22300,00	1013,64	32	32436,36
Аналітик систем обробки графічних даних	24500,00	1113,64	5	5568,18
Консультант-аналітик обробки цифрових зображень	24500,00	1113,64	7	7795,45
Провідний фахівець	8300,00	377,27	10	3772,73
Всього				78972,73

Розглянеом основну заробітну плату робітників. Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт НДР на тему «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі» розраховуємо за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i , \quad (5.3)$$

де C_i – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год; t_i – час роботи робітника при виконанні визначеної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду C_i можна визначити за формулою:

$$C_i = \frac{M_M \cdot K_i \cdot K_c}{T_p \cdot t_{зм}} , \quad (5.4)$$

де M_M – розмір мінімальної місячної заробітної плати, приймемо $Мм$=8000,00 грн; K_i – коефіцієнт міжкваліфікаційного співвідношення (табл. Б.2) [39, 40]; K_c – мінімальний коефіцієнт співвідношень місячних тарифних ставок; T_p – середнє число робочих днів в місяці, приблизно T_p = 22 дн; $t_{зм}$ – тривалість зміни, год. C_1 = 8000,00 · 1,10 · 1,15 / (22 · 8) = 57,50 *грн.* $З_{р1}$ = 57,50 · 8,00 = 460,00 *грн.*

Проведені розрахунки зведемо до таблиці 5.4.

Таблиця 5.4 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Встановлення допоміжного обладнання робочих місць дослідників	8,00	2	1,10	57,50	460,00
Інсталяція програмного забезпечення	7,50	3	1,35	70,57	529,26
Встановлення цифрових обчислювальних систем	5,60	4	1,50	78,41	439,09

Продовження таблиці 5.4

Найменування робіт	Тривалість роботи, год	Розряд роботи	Тарифний коефіцієнт	Погодинна тарифна ставка, грн	Величина оплати на робітника грн
Відлагодження інтерполяційних модулів графічної системи	6,35	5	1,70	88,86	564,28
Підготовка цифрової експериментальної моделі обробки зображень	5,00	5	1,70	88,86	444,32
Формування бази даних зображень	16,00	3	1,35	70,57	1129,09
Тренування системи	5,00	2	1,10	57,50	287,50
Всього					3853,55

Розглянемо детальніше додаткову заробітна плата дослідників та робітниківю Додаткову заробітну плату розраховуємо як 10 … 12% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод} = (З_о + З_р) \cdot \frac{Н_{дод}}{100\%}, \quad (5.5)$$

де $Н_{дод}$ – норма нарахування додаткової заробітної плати. Приймемо 10%. *$З_{дод}$ = (78972,73 + 3853,55) · 10 / 100% = 8282,63 грн.*

5.3.2 Відрахування на соціальні заходи

Нарахування на заробітну плату дослідників та робітників розраховуємо як 22% від суми основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (5.6)$$

де $Н_{зп}$ – норма нарахування на заробітну плату. Приймаємо 22%. *Зн = (78972,73 + 3853,55 + 8282,63) · 22 / 100% = 20043,96 грн.*

5.3.3 Сировина та матеріали

Витрати на матеріали (*М*), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$M = \sum_{j=1}^{n} H_j \cdot Ц_j \cdot K_j - \sum_{j=1}^{n} B_j \cdot Ц_{вj}\ , \qquad (5.7)$$

де H_j – норма витрат матеріалу *j*-го найменування, кг; *n* – кількість видів матеріалів; $Ц_j$ – вартість матеріалу *j*-го найменування, грн/кг; K_j – коефіцієнт транспортних витрат, (K_j = *1,1 ... 1,15*); B_j – маса відходів *j*-го найменування, кг; $Ц_{вj}$ – вартість відходів *j*-го найменування, грн/кг. M_1 = *3,0 · 212,00 · 1,1 - 0 · 0 = 699,60 грн.*

Проведені розрахунки зведемо до таблиці 5.5.

Таблиця 5.5 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
Папір канцелярський офісний (А4-500/80)	212,00	3,0	0	0	699,60
Папір для заміток (А5)-500/70	112,00	3,0	0	0	369,60
Начиння канцелярське Lezard Ultra	216,00	4,0	0	0	950,40
Органайзер офісний Lezard Ultra	260,00	3,0	0	0	858,00
Картридж для принтера Canon 750AF-DX	2120,00	2,0	0	0	4664,00
Диск оптичний LG-10 (CD-R)	23,00	5,0	0	0	126,50
Диск оптичний LG-W (CD-RW)	32,00	5,0	0	0	176,00

Продовження таблиці 5.5

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, кг	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн
USB-пам'ять Kingstar (32 ГБ) Class 10	189,00	1,0	0	0	207,90
USB-пам'ять Kingstar (64 ГБ) Class 10 A	290,00	2,0	0	0	638,00
Всього					8690,00

5.3.4 Розрахунок витрат на комплектуючі

Витрати на комплектуючі *($К_в$)*, які використовують при проведенні НДР на тему «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі», розраховуємо, згідно з їхньою номенклатурою, за формулою:

$$К_в = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j \qquad (5.8)$$

де $Н_j$ – кількість комплектуючих j-го виду, шт.; $Ц_j$ – покупна ціна комплектуючих j-го виду, грн; $К_j$ – коефіцієнт транспортних витрат, ($К_j$ *= 1,1 ... 1,15*). $К_в = 1 \cdot 3152{,}00 \cdot 1{,}1 = 3467{,}20$ *грн.*

Проведені розрахунки зведемо до таблиці 5.6.

Таблиця 5.6 – Витрати на комплектуючі

Найменування комплектуючих	Кількість, шт.	Ціна за штуку, грн	Сума, грн
Мережева карта Mikrotik R11e-LoRa	1	3152,00	3467,20
Відеокарта ASUS GeForce RTX3060 12Gb DUAL OC V2 LHR (DUAL-RTX3060-O12G-V2)	1	12299,00	13528,90
Кабелі інтерфейсів	3	850,00	2805,00

Всього	19801,10

5.3.5 Спецустаткування для наукових (експериментальних) робіт

Балансову вартість спецустаткування розраховуємо за формулою:

$$B_{спец} = \sum_{i=1}^{k} Ц_i \cdot C_{пр.i} \cdot K_i \ , \qquad (5.9)$$

де $Ц_i$ – ціна придбання одиниці спецустаткування даного виду, марки, грн; $C_{пр.i}$ –кількість одиниць устаткування відповідного найменування, які придбані для проведення досліджень, шт.; K_i – коефіцієнт, що враховує доставку, монтаж, налагодження устаткування тощо, (K_i = *1,10…1,12);* k – кількість найменувань устаткування. $B_{спец} = 18960{,}00 \cdot 1 \cdot 1{,}1 = 20856{,}00$ *грн.*

Отримані результати зведемо до таблиці 5.7.

Таблиця 5.7 – Витрати на придбання спецустаткування по кожному виду

Найменування устаткування	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Мультимедійна проекційна система	1	18960,00	20856,00
Монітор Samsung LC27G55TQBIXCI 4K	1	10999,00	12098,90
Графічний планшет Wacom Intuos S Black (CTL-4100K-N)	1	4200,00	4620,00
Всього			37574,90

5.3.6 Програмне забезпечення для наукових (експериментальних) робіт

Балансову вартість програмного забезпечення розраховуємо за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (5.10)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн; $C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які

придбані для проведення досліджень, шт.; K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, (K_i = *1,10…1,12);* k – кількість найменувань програмних засобів. $B_{прг} = 1380,00 \cdot 1 \cdot 1,1 = 1518,00$ *грн.*

Отримані результати зведемо до таблиці 5.8.

Таблиця 5.8 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
IntelliJ IDEA для розробки на Java	1	1380,00	1518,00
MathType	1	7860,00	8646,00
Прикладний пакет Microsoft Visual Studio Community 2019	1	7646,00	8410,60
Середовище програмування мови програмування – C/C++	1	4850,00	5335,00
База даних тестових зображень у форматі .BMP	1	4350,00	4785,00
Всього			28694,60

5.3.7 Амортизація обладнання, програмних засобів та приміщень

У спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, розраховуємо з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (5.11)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн; $t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців; $T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

$A_{обл}$ = *(45389,00 · 2) / (3 · 12) = 2521,61 грн.*

Проведені розрахунки зведемо до таблиці 5.9.

Таблиця 5.9 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Електронний комплекс аналітичної системи ПК DELL OPTIPLEX 7010 MFF / I5-13500T (N007O7010MFF)	45389,00	3	2	2521,61
Персональний комп'ютер HP PRODESK 405 G6 SFF / RYZEN3 4300G (294D5EA)	19699,00	3	2	1094,39
Спеціалізоване робоче місце дослідника	8699,00	5	2	289,97
Пристрій виводу текстової інформації	6899,00	5	2	229,97
Оргтехніка	8400,00	5	2	280,00
Приміщення лабораторії досліджень	399000,00	25	2	2660,00
ОС Windows 11	6500,00	3	2	361,11
Прикладний пакет Microsoft Office 2019	6540,00	3	2	363,33
Всього				7800,38

5.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію (B_e) розраховуємо за формулою:

$$B_e = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_e \cdot K_{вni}}{\eta_i}, \quad (5.12)$$

де W_{yi} – встановлена потужність обладнання на визначеному етапі розробки, кВт; t_i – тривалість роботи обладнання на етапі дослідження, год; $Ц_e$ – вартість 1 кВт-години електроенергії, грн; приймемо Ц$_e$ = 10,98 грн; $K_{вni}$ – коефіцієнт, що враховує використання потужності, $K_{вni} < 1$; η_i – коефіцієнт корисної дії обладнання, $\eta_i < 1$. $B_e = 0,32 \cdot 220,0 \cdot 10,98 \cdot 0,95 / 0,97 = 772,99$ *грн.*

Проведені розрахунки зведемо до таблиці 5.10.

Таблиця 5.10 – Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
Електронний комплекс аналітичної системи ПК DELL OPTIPLEX 7010 MFF / I5-13500T (N007O7010MFF)	0,32	220,0	772,99
Персональний комп'ютер HP PRODESK 405 G6 SFF / RYZEN3 4300G (294D5EA)	0,25	220,0	603,90
Спеціалізоване робоче місце дослідника	0,08	220,0	193,25
Пристрій виводу текстової інформації	0,25	4,5	12,35
Оргтехніка	0,50	2,0	10,98
Мультимедійна проекційна система	0,23	100,0	252,54
Монітор Samsung LC27G55TQBIXCI	0,06	100,0	65,88
Всього			1911,89

5.3.9 Службові відрядження

Витрати за статтею «Службові відрядження» відсутні.

5.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати розраховуємо як 30…45% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{сп} = (З_о + З_р) \cdot \frac{Н_{сп}}{100\%}, \quad (5.13)$$

де $Н_{сп}$ – норма нарахування за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації», приймемо $Н_{сп}$= 30%. *$B_{сп}$ = (78972,73 + 3853,55) · 30 / 100% = 24847,88 грн.*

5.3.11 Інші витрати

Витрати за статтею «Інші витрати» розраховуємо як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$І_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%}, \quad (5.14)$$

де *$Н_{ів}$* – норма нарахування за статтею «Інші витрати», приймемо $Н_{ів}$ = 50%. *$І_в$ = (78972,73 + 3853,55) · 50 / 100% = 41413,14 грн.*

5.3.12 Накладні (загальновиробничі) витрати

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуємо як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$B_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \quad (5.15)$$

де $H_{нзв}$ – норма нарахування за статтею «Накладні (загальновиробничі) витрати», приймемо $Н_{нзв}$ = 100%. *$В_{нзв}$ = (78972,73 + 3853,55) · 100 / 100% = 82826,27 грн.*

Витрати на проведення науково-дослідної роботи на тему «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі» розраховуємо як суму всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (5.16)$$

$В_{заг}$= 78972,73 +3853,55 +8282,63 +20043,96 +8690,00 +19801,10 +37574,90 + 28694,60 + 7800,38 +1911,89 +0,00 +24847,88 +41413,14 +82826,27 = 364713,02 грн.

Загальні витрати *ЗВ* на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховується за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (5.17)$$

де η - коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи, приймемо η=0,9. *ЗВ = 364713,02 / 0,9 = 405236,69 грн.*

5.4 Розрахунок економічної ефективності науково-технічної розробки при її можливій комерціалізації потенційним інвестором

Результати дослідження проведені за темою «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі» передбачають комерціалізацію протягом 4-х років реалізації на ринку.

У зазначеному випадку майбутній економічний ефект буде формуватися на основі таких даних: ΔN – збільшення кількості споживачів продукту, у періоди часу, що аналізуються, від покращення його певних характеристик (див. таб 5.11); N – кількість споживачів які використовували аналогічний продукт у році до впровадження результатів нової науково-технічної розробки, приймемо

6800 осіб; $Ц_б$ – вартість програмного продукту у році до впровадження результатів розробки, приймемо 2860,00 грн; $\pm\Delta Ц_о$ – зміна вартості програмного продукту від впровадження результатів науково-технічної розробки, приймемо 252,62 грн.

Таблиця 5.11 – Показники на 4 роки

Показник	1-й рік	2-й рік	3-й рік	4-й рік
Збільшення кількості споживачів, осіб	450	600	950	700

Можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із 4-х років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховуємо за формулою [39, 40]**:**

$$\Delta П_i = (\pm\Delta Ц_о \cdot N + Ц_о \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\vartheta}{100}), \quad (5.18)$$

де λ – коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2024 році ставка податку на додану вартість складає 20%, а коефіцієнт λ *=0,8333*; ρ – коефіцієнт, який враховує рентабельність інноваційного продукту). Приймемо ρ =40%; ϑ – ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2024 році ϑ *=18%*;

Збільшення чистого прибутку 1-го року:

$\Delta П_1$ =(252,62·6800,00+3112,62·450)·0,83·0,4·(1-0,18/100%) = 848979,08 грн.

Збільшення чистого прибутку 2-го року:

$\Delta П_2$ =(252,62·6800,00+3112,62·1050)·0,83·0,4·(1-0,18/100%) = 1357406,88 грн.

Збільшення чистого прибутку 3-го року:

$\Delta П_3$ =(252,62·6800,00+3112,62·2000)·0,83·0,4·(1-0,18/100%) = 2162417,57 грн.

Збільшення чистого прибутку 4-го року:

$\Delta П_4$ =(252,62·6800,00+3112,62·2700)·0,83·0,4·(1-0,18/100%) = 2755583,33 грн.

Приведена вартість збільшення всіх чистих прибутків *ПП*, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (5.19)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн; T – період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки; τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ =0,14; t – період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = 848979,08/(1+0,14)^1+1357406,88/(1+0,14)^2+2162417,57/(1+0,14)^3+$$
$$+2755583,33/(1+0,14)^4=744718,49+1044480,52+1459570,26+1631526,54=$$
$$= 4880295,81 \text{ грн.}$$

Величина початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки:

$$PV = k_{інв} \cdot ЗВ, \quad (5.20)$$

де $k_{інв}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію, приймаємо $k_{інв}$ =2; *ЗВ* – загальні витрати на проведення науково-технічної розробки та оформлення її результатів, приймаємо 405236,69 грн. $PV = k_{інв} \cdot ЗВ$ = 2 · 405236,69 = 810473,38 грн.

Абсолютний економічний ефект $E_{абс}$ для потенційного інвестора від можливого впровадження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV, \quad (5.21)$$

де $ПП$ – приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, 4880295,81 грн; PV – теперішня вартість початкових інвестицій, 810473,38 грн. $E_{абс} = ПП - PV$ = 4880295,81 - 810473,38 = 4069822,44 грн.

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1, \quad (5.22)$$

де $E_{абс}$ – абсолютний економічний ефект вкладених інвестицій, 4069822,44 грн; PV – теперішня вартість початкових інвестицій, 810473,38 грн; $T_ж$ – життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впровадження, 4 роки. $E_в = \sqrt[Tж]{1 + \frac{E_{абс}}{PV}} - 1$ = (1+4069822,44/810473,38)$^{1/4}$= 0,57.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$:

$$\tau_{мін} = d + f, \quad (5.23)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d =0,11; f – показник, що характеризує ризикованість вкладення інвестицій, приймемо 0,3. $\tau_{мін}$= 0,11+0,3 = 0,41 < 0,57 свідчить про те, що внутрішня економічна дохідність інвестицій $E_в$, вища мінімальної внутрішньої дохідності. Тобто інвестувати в науково-дослідну роботу за темою «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі» доцільно.

Період окупності інвестицій $T_{ок}$ які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (5.24)$$

де $E_в$ – внутрішня економічна дохідність вкладених інвестицій. $T_{ок}$ = 1 / 0,57 = 1,77 р. $T_{ок}$ < 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

5.3 Висновки

Згідно проведених досліджень рівень комерційного потенціалу розробки за темою «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі» становить 39,3 бала, що, свідчить про комерційну важливість проведення даних досліджень (рівень комерційного потенціалу розробки вище середнього).

При оцінюванні за технічними параметрами, згідно узагальненого коефіцієнту якості розробки, науково-технічна розробка переважає існуючі аналоги приблизно в 2,42 рази.

Також термін окупності становить 1,77 р., що менше 3-х років, що свідчить про комерційну привабливість науково-технічної розробки і може спонукати потенційного інвестора профінансувати впровадження даної розробки та виведення її на ринок.

Отже можна зробити висновок про доцільність проведення науково-дослідної роботи за темою «Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі».

ВИСНОВКИ

У ході виконання магістерської кваліфікаційної роботи було розроблено інформаційну технологію для оптимізації побудови зображень на шестикутному растрі, що відповідає сучасним вимогам у галузі графічної візуалізації, обробки зображень та комп'ютерної графіки. У дослідженні було реалізовано декілька важливих етапів, кожен з яких сприяв створенню якісного, інноваційного програмного забезпечення.

На основі детального аналізу сучасних методів формування графічних примітивів і порівняння з існуючими технологіями, було обґрунтовано переваги використання шестикутного растру над традиційним квадратним. Завдяки його структурним особливостям досягається рівномірніше покриття площини, мінімізація артефактів і підвищення якості відображення криволінійних форм. Це робить шестикутний растр перспективним для використання в задачах, що потребують високої точності та продуктивності.

Розроблено нові алгоритми та методики для роботи з шестикутним растром, включаючи удосконалення методу побудови прямих, кругової інтерполяції та антиаліайзингу. Ці алгоритми дозволяють досягти високої точності та реалістичності графічних зображень, а також значно зменшити обчислювальні витрати під час їх обробки.

Було виконано вибір технологій та розроблено архітектуру програмного забезпечення, що забезпечує багатоплатформність, масштабованість та адаптивність. Використання мови програмування Java, технології JavaFX для створення інтерфейсу користувача та сучасних підходів до моделювання дозволило реалізувати систему, яка задовольняє сучасні потреби користувачів і відповідає високим стандартам розробки програмного забезпечення.

Особливу увагу було приділено розробці графічного інтерфейсу, який відповідає сучасним вимогам ергономіки, інтуїтивності та функціональності. Структура інтерфейсу забезпечує зручну роботу з налаштуваннями параметрів і візуалізацією результатів у режимі реального часу.

Ключовим досягненням стало створення модуля конвертації зображень у формат гексагонального растру, який інтегрує процес завантаження зображень, налаштування параметрів, динамічний попередній перегляд та експорт результатів у різних форматах. Архітектура модуля дозволяє використовувати його як автономний інструмент або в складі більш складних систем.

Проведений економічний аналіз підтвердив доцільність розробки і високий комерційний потенціал системи, що створює можливості для її впровадження та комерціалізації. Рівень узагальненого коефіцієнта якості розробки перевищує існуючі аналоги, а термін окупності свідчить про привабливість проекту для потенційних інвесторів.

Результати дослідження відкривають перспективи для подальшого вдосконалення і розширення функціоналу програмного забезпечення, а також його використання в різноманітних галузях, таких як аналіз даних, обробка графіки та комп’ютерний дизайн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Романюк О. Н. Панфілова Ю. О. Деякі застосування гексагональної моделі піксела. Інформаційно-комп'ютерні технології – 2020 : тези доп. XI Міжнародної науково-технічної конференції, м. Житомир, 09 – 11 квітня 2020 р. / Житомирська політехніка, 2020. – С. 116–117.

2. Комп'ютерна програма для імітації гексагонального растру / О. Н. Романюк., В. А. Шмалюх, О. В. Мельник, А. В. Марущак. *Інформаційні технології в освіті, техніці та промисловості* : тези Республ. наук.-практ. конф. 2020 р. Івано-Франківськ. С. 70–71.

3. Романюк О. Н., Шмалюх В. А., Марущак А. В. Особливості інтегрованої графіки. *Інформаційні технології в культурі, мистецтві, освіті, науці, економіці та бізнесі* : матеріали V Міжнар. науково-практ. конф, м. Київ, 22 квіт. 2020 р. Київ, 2020. С. 331–333.

4. Шмалюх В. А., Романюк О. В. Аналіз підходів до тестування зручності використання програмних додатків. *Електронні інформаційні ресурси: створення, використання, пам'яті Олексія Петровича Стахова* : Зб. матеріалів Міжнар. науково-практ. Інтернет конф., м. Суми/Вінниця, 9 листоп. 2021 р. Вінниця, 2021. С. 220–223.

5. Пакети статистичних програм для аналізу та обробки даних / О. Н. Романюк, В. А. Шмалюх, А. В. Марущак, О. М. Ціхановська *Нові інформаційні технології управління бізнесом* : Зб. тез IV Всеукр. науково-практ. конф., м. Київ. Київ, 2021. С. 405–409.

6. Аналіз технології SLI об'єднання відеокарт / В. А. Шмалюх та ін. *The 2 nd International scientific and practical conference "Fundamental and applied research in the modern world"* : матеріали міжнар. наук.-практ. конф., м. Boston, 23–25 верес. 2020 р. Boston, 2020. С. 507–513.

7. Розробка програм для тренування операторів БПЛА / В. А. Шмалюх та ін. *L Науково-технічна конференція факультету інформаційних технологій та*

комп'ютерної інженерії : матеріали наук.-практ. конф., м. Вінниця, 18 берез. 2021 р. Вінниця, 2021. С. 3.

8. Технологія HDR для моніторів / В. А. Шмалюх та ін. *Електронні інформаційні ресурси: створення, використання, доступ* : матеріали міжнар. наук.-практ. конф., Віннця, 9–10 листоп. 2020 р. Вінниця, 2020. С. 163–168.

9. Формування графічних зображень шляхом масштабуваня відеокарт : монографія / В. А. Шмалюх та ін. 2-ге вид. Одеса : КУПРІЄНКО СВ, 2020. Т. 2 : Рівень розвитку техніки і технологій в XXI столітті.

10. Шмалюх В. А., Марущак А. В., Коваленко О. О. Розвиток стандартів створення програмних продуктів в світі та в Україні. *Електронні інформаційні ресурси: створення, використання, доступ* : Зб. матеріалів Міжнар. науково-практ. Інтернет конф., м. Суми/Вінниця, 9–10 листоп. 2020 р. Суми/Вінниця, 2020. С. 126–131. URL: http://ir.lib.vntu.edu.ua//handle/123456789/31436.

11. Аналіз стандарту AVIF стиснення графічного зображення / О. Н. Романюк, В. А. Шмалюх, А. В. Марущак, В. П. Майданюк *Матеріали Республіканської науково-практичної конференції «Інформаційні технології в освіті, техніці та промисловості»* : матеріали міжнар. наук.-практ. конф., м. Івано-Франківськ, 8 жовт. 2020 р. Івано-Франківськ, 2020. С. 64–65.

12. Аналіз технології SLI об’єднання відеокарт / О. Н. Романюк, В. А. Шмалюх, А. В. Марущак, К. О. Корнієнко *The 2 nd International scientific and practical conference “Fundamental and applied research in the modern world”* : матеріали міжнар. наук.-практ. конф., м. Boston, 23–25 верес. 2020 р. Boston, 2020. С. 507–513. URL: http://ir.lib.vntu.edu.ua//handle/123456789/30705 (дата звернення: 10.11.2024).

13. Шмалюх В. А., Марущак А. В., Коваленко О. О. Формування команди та застосування методології Scrum. *L Науково-технічна конференція факультету інформаційних технологій та комп'ютерної інженерії* : матеріали наук.-практ. конф., м. Вінниця, 18 берез. 2021 р. Вінниця, 2021. С. 5.

14. Шмалюх В. А., Марущак А. В., Романюк О. Н. Шейдери та їх мови програмування. *Modern science, practice, society* : Abstracts of XIX International

Scientific and Practical Conference, м. Boston, 25–26 трав. 2020 р. Boston, 2020. С. 402–406.

15. Шмалюх В. А., Марущак А. В., Романюк О. Н. Аналіз архітектури RNDA. *The 9th International scientific and practical conference "Scientific achievements of modern society"* : матеріали міжнар. наук.-практ. конф., м. Liverpool, 28–30 квіт. 2020 р. Liverpool, 2020. С. 4–10.

16. Шмалюх В. А., Романюк О. Н., Мельник О. В. Розробка застосунку для формування відрізків прямих на гексагональному растрі. *Молодь в науці: дослідження, проблеми, перспективи* : матеріали Всеукраїнської науково-практичної інтернет-конференції, м. Вінниця, 2022 р. Вінниця : ВНТУ, 2022.

17. Романюк О. Н., Мельник О. В., Шмалюх В. А. Метод прискореної кругової інтерполяції на гексагональному растрі. Вісник Вінницького політехнічного інституту. 2023. № 2. С. 81–88.

18. Романюк О. Н., Мельник О. В., Шмалюх В. А., Чехместрук Р. Ю. Програмний модуль для формування кіл на гексагональному растрі. *Modern Research in World Science* : матеріали XII Міжнародної науково-практичної конференції, м. Львів, 26–28 лют. 2023 р. Львів, 2023.

19. Шмалюх В. А., Козловський О. С., Богач І. В. Розробка бекенд-систем на базі монолітної архітектури з перспективою переходу до мікросервісної архітектури // *Молодь в науці: дослідження, проблеми, перспективи* : матеріали Всеукраїнської науково-практичної інтернет-конференції, м. Вінниця, 2 грудня 2024 р. Вінниця: ВНТУ, 2024.

20. Шмалюх В. А., Богач І. В. Реалізація інформаційної технології для оптимізації побудови зображень на шестикутному растрі // *Молодь в науці: дослідження, проблеми, перспективи :* матеріали Всеукраїнської науково-практичної інтернет-конференції, м. Вінниця, 2 грудня 2024 р. Вінниця: ВНТУ, 2024.

21. Романюк О. Н., Мельник О. В. Особливості використання гексагонального растра при побудові пристроїв відображення. Вимірювальна та обчислювальна техніка в технологічних процесах. 2016. № 3. С. 105-109.

22. Романюк О. Н., Мельник О. В. Особливості гексагональної моделі піксела. Вимірювальна та обчислювальна техніка в технологічних процесах. 2014. № 1. С. 91-95.

23. Романюк О. Н., Мельник О. В., Коваль Л. Г. Використання гексагональних комірок у видавничій справі. Інформація, комунікація та управління знаннями в глобалізованому світі Матеріали П'ятої міжнародної наукової конференції «Інформація, комунікація та управління знаннями в глобалізованому світі», Київ, 22 травня, 2022. С.45-47.

24. Романюк О. Н., Мельник О. В. Особливості використання гексагонального растра при побудові пристроїв відображення. Вимірювальна та обчислювальна техніка в технологічних процесах. 2016. № 3. С. 105-109.

25. Панфілова Ю. О., Романюк О Н., Мельник О. В. Використання гексагонального растру в комп'ютерних іграх. Інформаційно-комп'ютерні технології: тези доп. XII Міжнародної науково-технічної конференції, м. Житомир, 01 - 03 квітня 2021 р. / Житомирська політехніка, 2021. С.205.

26. Пєтух А. М., Обідник Д. Т., Романюк О.Н. Інтерполяція в задачах контурного формоутворення: монографія. Вінниця: ВНТУ, 2007. С.103 .

27. INKSCAPE Draw Freely. *Inkscape*. URL: https://inkscape.org (дата звернення: 10.11.2024).

28. Processing is a flexible software sketchbook and a language for learning how to code. *Processing*. URL: https://processing.org (дата звернення: 10.11.2024).

29. Marmoset hexels grid-based vector art, pixel art, design & animation. *Hexels*. URL: https://marmoset.co/hexels (дата звернення: 10.11.2024).

30. Романюк О. Н. Панфілова Ю. О. Деякі застосування гексагональної моделі піксела. Інформаційно-комп'ютерні технології – 2020 : тези доп. XI Міжнародної науково-технічної конференції, м. Житомир, 09 – 11 квітня 2020 р. / Житомирська політехніка, 2020. – С. 116–117.

31. Романюк О. Н., Мельник О. В. Формування відрізків прямих на гексагональному растрі. Наукові праці Донецького національного технічного

університету. Серія «Інформатика, кібернетика та обчислювальна техніка». 2016. №2(23). – С. 69-72.

32. Романюк О. Н., Мельник О. В., Романюк О. В. Реалізація кругової інтерполяції при використанні гексагонального растру. Наукові праці Донецького національного технічного університету. Серія : Інформатика, кібернетика та обчислювальна техніка. 2017. № 1. – С. 53-58.

33. Романюк О. Н., Мельник О. В., Шмалюх В. А. Метод прискореної кругової інтерполяції на гексагональному растрі. Вісник Вінницького Політехнічного Інституту. 2023. № 2 (167). – С. 9-18.

34. Романюк О. Н., Курінний М. С., Мельник О. В. Антиаліайзинг зображення відрізків прямих з використанням нової моделі піксела. *Методи та системи оптико-електронної і цифрової обробки зображень та сигналів*. 2015. С. 30–35.

35. Get Java for desktop applications Download Java. *java*. URL: https://www.java.com/en/ (дата звернення: 10.11.2024).

36. IntelliJ IDEA – the Leading Java and Kotlin IDE URL: https://www.jet-brains.com/idea/ (дата звернення: 10.11.2024).

37. Колір у графічному дизайні та рекламі URL: https://koloris-tika.kiev.ua/t_vkd.php (дата звернення: 10.11.2024).

38. Види тестування та відмінності між ними URL: https://qagroup.com.ua/publications/vydy-testuvannya-ta-vidminnosti-mizh-nymy/ (дата звернення: 10.11.2024).

39. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

40. Кавецький В. В. Економічне обґрунтування інноваційних рішень: практикум / В. В. Кавецький, В. О. Козловський, І. В. Причепа – Вінниця : ВНТУ, 2016. – 113 с.

ДОДАТКИ

Додаток А Технічне завдання (обов’язковий)

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«14» 10 2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
«Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі»
08-31.МКР.006.02.000

Керівник: к.т.н., доц. каф. АІІТ
Ілона БОГАЧ
«11» 10 2024 р.
Розробив студент гр. 1ІСТ-23М
Владислав ШМАЛЮХ
«11» 10 2024 р.

Вінниця
ВНТУ 2024

1. Назва та галузь застосування

Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі. Галузь застосування: комп’ютерна графіка, геоінформаційні системи, системи візуалізації, аналітичні інструменти та обробка даних.

2. Підстава для розробки

Підставою для виконання роботи є наказ №310 по ВНТУ від 17. 09 2024 р.

3. Мета та призначення розробки

Метою роботи є створення інформаційної технології, що забезпечує оптимізоване формування та відображення зображень на основі гексагонального растру. Призначенням є розробка програмного модуля для ефективної візуалізації графічних об’єктів із застосуванням гексагональної структури, яка підвищує якість зображення та усуває артефакти традиційної піксельної сітки.

4. Джерела розробки

1) Шмалюх В. А., Романюк О. Н., Мельник О. В. Розробка застосунку для формування відрізків прямих на гексагональному растрі // Молодь в науці: дослідження, проблеми, перспективи. – Вінниця: ВНТУ, 2022.

2) Романюк О. Н., Мельник О. В., Шмалюх В. А. Метод прискореної кругової інтерполяції на гексагональному растрі // Вісник Вінницького політехнічного інституту. – 2023. – № 2. – С. 81–88.

3) Романюк О. Н., Мельник О. В., Шмалюх В. А., Чехместрук Р. Ю. Програмний модуль для формування кіл на гексагональному растрі // Modern Research in World Science. – Львів, 2023.

5. Показники призначення

Основні технічні вимоги та мінімальні системні вимоги до програми: ОС

Windows 11/Ubuntu 20.04/macOS Ventura, процесор Intel Core i5/AMD Ryzen 5/Apple M2, оперативна пам’ять 8 GB, місце на диску 100 MB.

Методи дослідження: у роботі використовуються моделювання, оптимізація алгоритмів, аналітичні методи, експериментальне тестування.
Результати роботи програми: формування зображень на гексагональному растрі; формування відрізків, кіл і геометричних примітивів; попередній перегляд зображення у реальному часі; конвертація зображення із квадратного у шестикутний растр та експорт результатів.

6. Економічні показники
До економічних показників входять:
- витрати на розробку не більше 405 тис. грн.;
- приведена вартість прибутку за 4 роки 4,88 млн. грн.;
- мінімальна дохідність не менше 57%;
- термін окупності не більше 2 років.

7. Стадії розробки
а) Аналіз предметної області та системи, що слід оптимізувати _до 22.09.24.
б) Вибір інструментів розробки _до 09.10.24.
в) Дизайн архітектури та імплементація компонентів системи _до 30.10.24.
г) Аналіз та оптимізація системи _до 05.11.24.
д) Економічна частина _до 20.11.24.
е) Оформлення матеріалів до захисту МКР _до 30.11.24.

8. Порядок контролю та приймання
Рубіжний контроль провести до 15.11.2024 р.
Попередній захист МКР провести до 30.05.12.2024р.
Захист МКР провести до 10.12.2024р.
Розробив студент групи 1ІСТ-23м Владислав Шмалюх

Додаток Б (обов’язковий)

Графічна частина

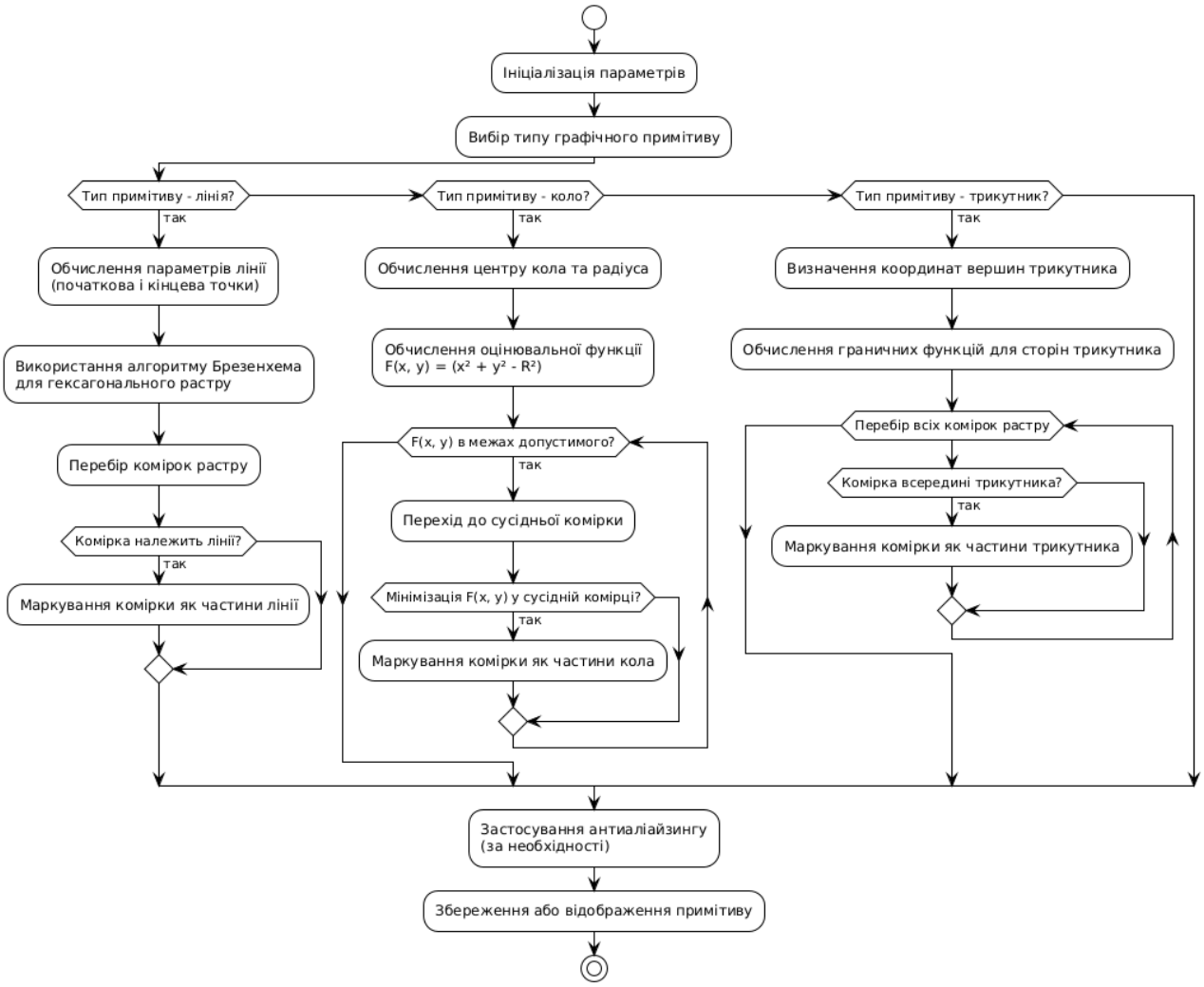

Рисунок Б.1 – Схема алгоритму формування базових графічних примітивів

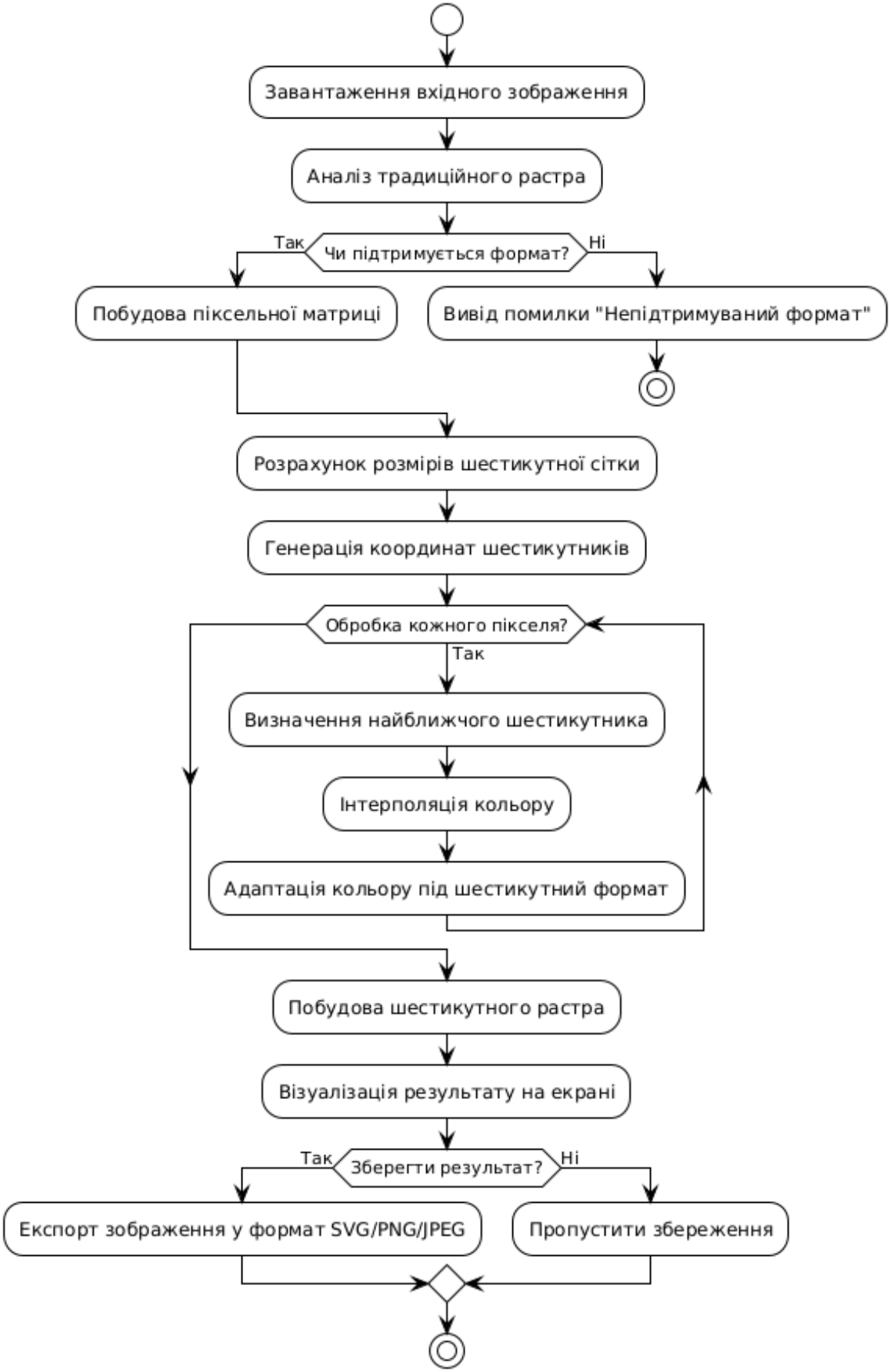

Рисунок Б.2 – Схема потоку даних для конвертації зображень

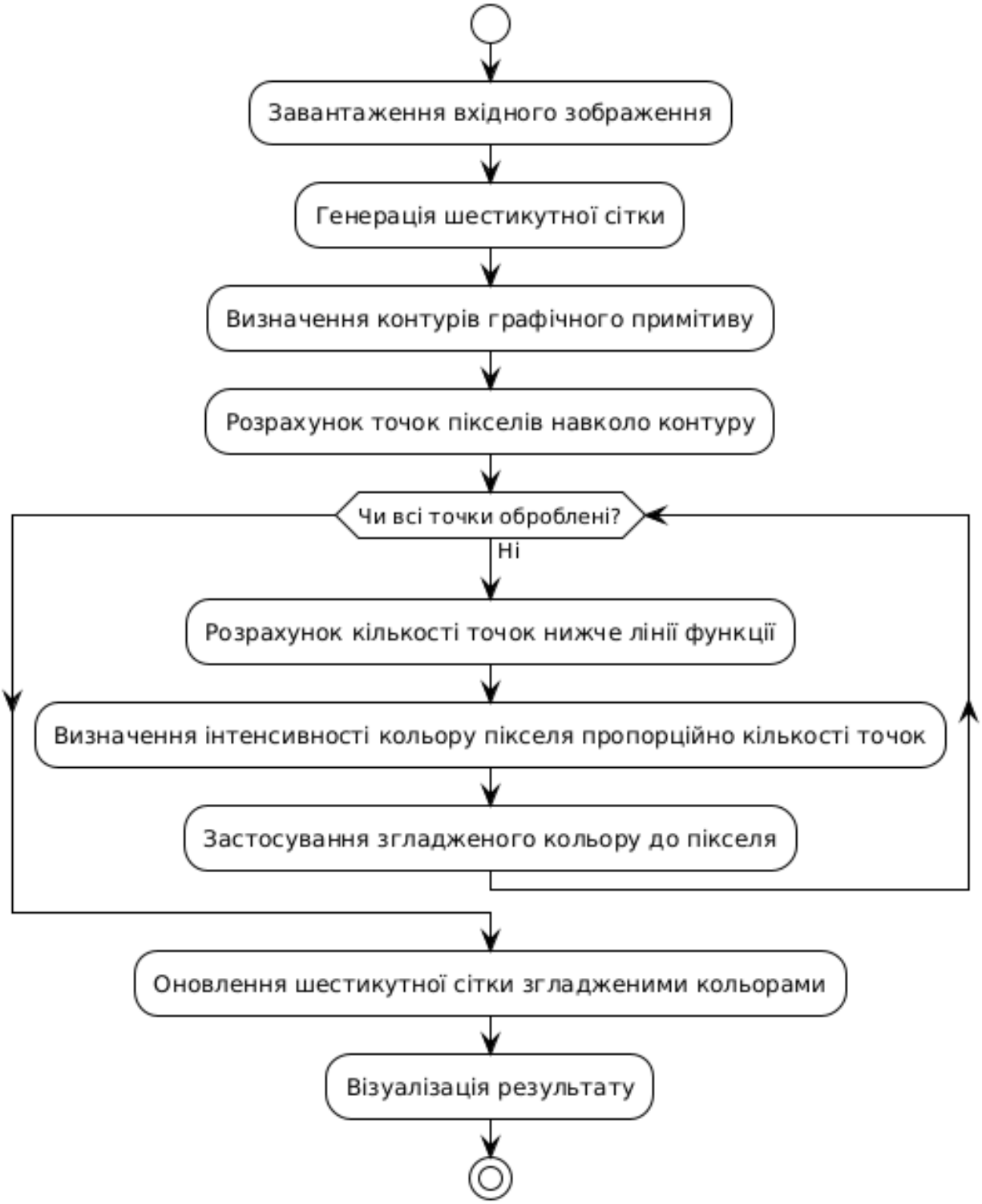

Рисунок Б.3 – Схема роботи антиаліайзингу на шестикутному растрі

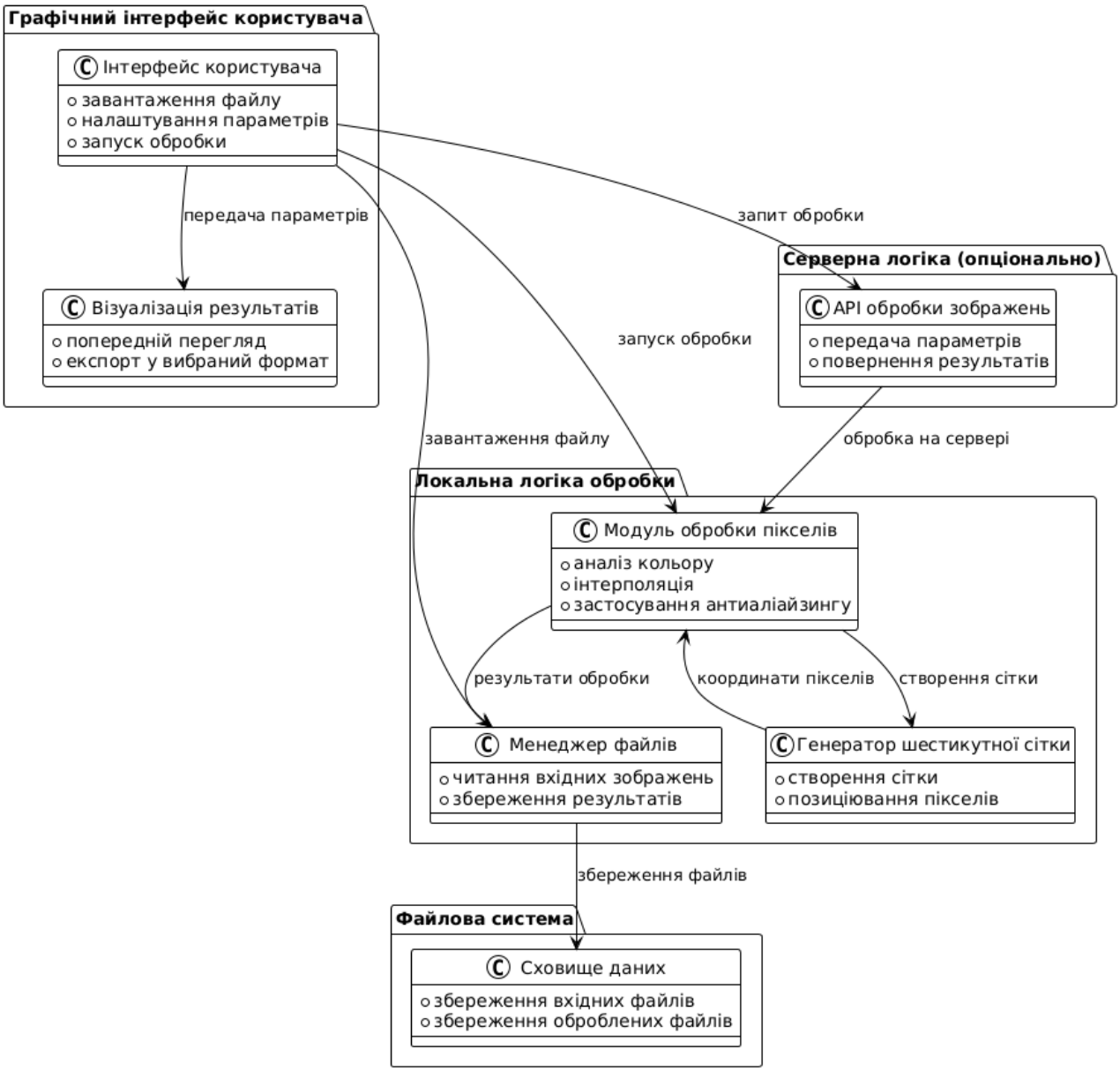

Рисунок Б.4 – Схема архітектури програмного модуля конвертації

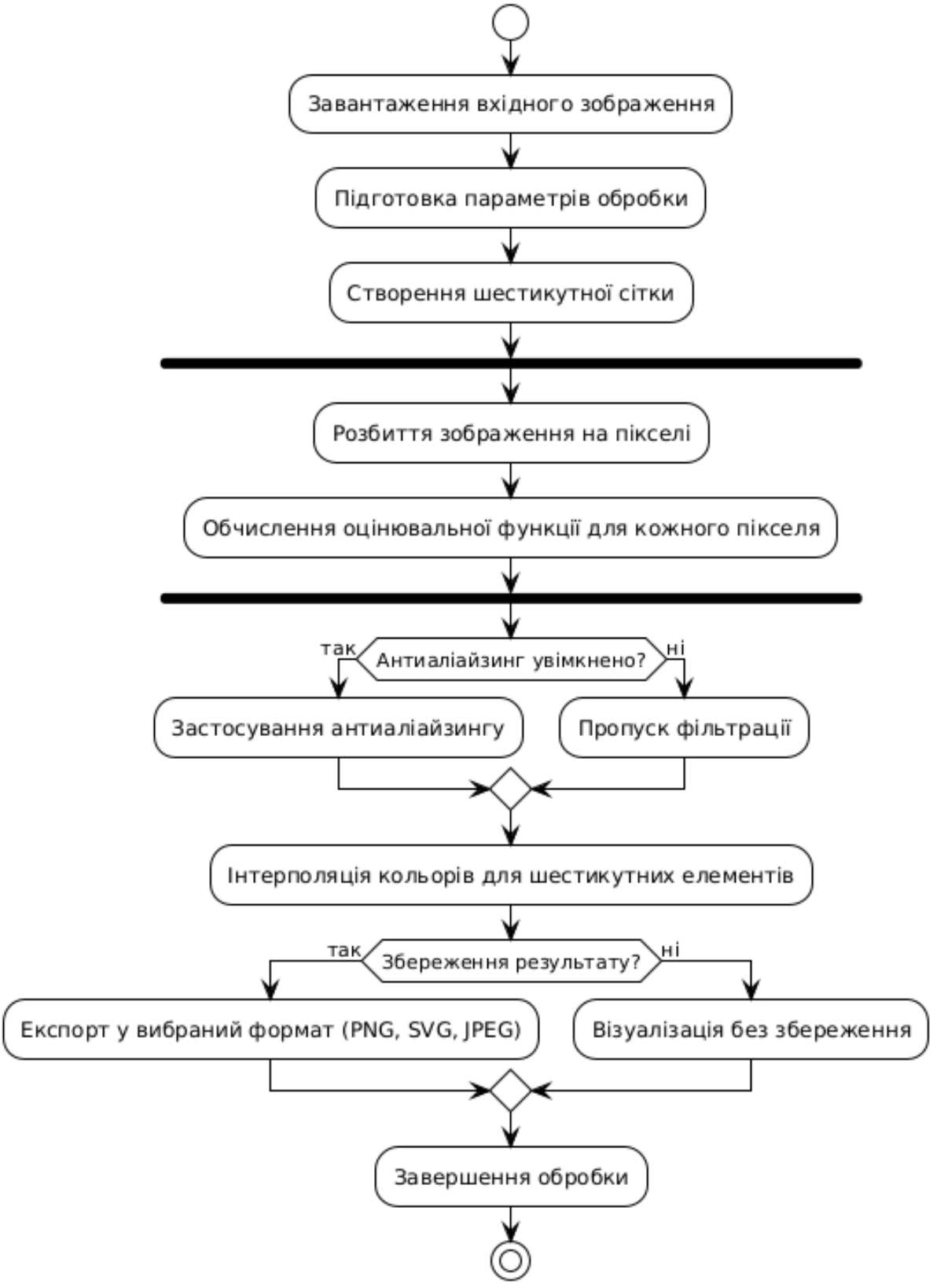

Рисунок Б.5 – Схема активності побудови складних зображень

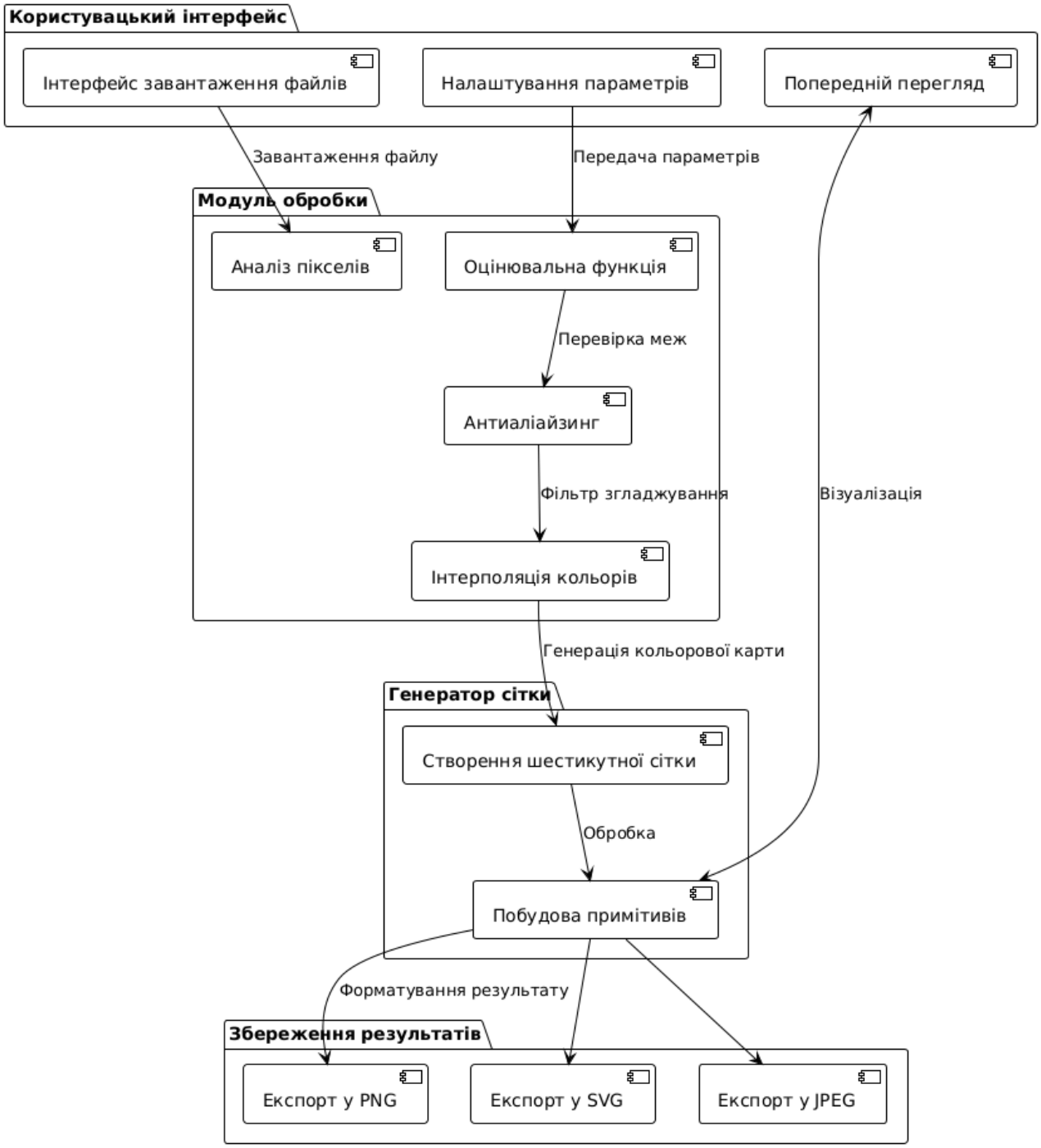

Рисунок Б.6 – Схема узгодження та взаємодії компонентів модулю конвертації

Додаток В (обов’язковий)

Лістинг основних модулів

```
import lombok.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

/**
 * Represents a hexagonal coordinate in a cube-coordinate system.
 * Coordinates are defined by q, r, and s where q + r + s = 0.
 */
@Getter
@Setter
@AllArgsConstructor
@ToString
@EqualsAndHashCode
public class Hex {

    private static final Logger logger = LoggerFactory.getLogger(Hex.class);

    private final int q; // Horizontal coordinate
    private final int r; // Diagonal coordinate
    private final int s; // Vertical coordinate (calculated as -q - r)

    /**
     * Validates the cube coordinates (q, r, s).
     * Ensures that q + r + s == 0.
     */
    public Hex {
        if (q + r + s != 0) {
            logger.error("Invalid Hex coordinates: q + r + s must equal 0. Provided values: q={}, r={}, s={}", q, r, s);
            throw new IllegalArgumentException("Invalid Hex coordinates: q + r + s must equal 0.");
        }
        logger.info("Hex created: {}", this);
    }

    /**
     * Adds two Hex coordinates.
     *
     * @param other the Hex to add.
     * @return a new Hex resulting from the addition.
     */
    public Hex add(Hex other) {
        Hex result = new Hex(this.q + other.q, this.r + other.r, this.s + other.s);
        logger.debug("Adding Hex {} to Hex {}. Result: {}", this, other, result);
        return result;
    }

    /**
```

```
     * Subtracts another Hex from this one.
     *
     * @param other the Hex to subtract.
     * @return a new Hex resulting from the subtraction.
     */
    public Hex subtract(Hex other) {
        Hex result = new Hex(this.q - other.q, this.r - other.r, this.s - other.s);
        logger.debug("Subtracting Hex {} from Hex {}. Result: {}", other, this, result);
        return result;
    }

    /**
     * Scales this Hex by a given factor.
     *
     * @param factor the scaling factor.
     * @return a new Hex scaled by the factor.
     */
    public Hex scale(int factor) {
        Hex result = new Hex(this.q * factor, this.r * factor, this.s * factor);
        logger.debug("Scaling Hex {} by factor {}. Result: {}", this, factor, result);
        return result;
    }

    /**
     * Calculates the distance between this Hex and another Hex.
     *
     * @param other the other Hex.
     * @return the distance as an integer.
     */
    public int distanceTo(Hex other) {
        int distance = (Math.abs(this.q - other.q) + Math.abs(this.r - other.r) + Math.abs(this.s - other.s)) / 2;
        logger.debug("Calculating distance from Hex {} to Hex {}. Distance: {}", this, other, distance);
        return distance;
    }
}
/**
 * Represents a fractional hexagonal coordinate for precise calculations.
 * Coordinates are defined in cube format (q, r, s), but can include fractional values.
 */
@Getter
@Setter
@AllArgsConstructor
@ToString
@EqualsAndHashCode
public class FractionalHex {

    private static final Logger logger = LoggerFactory.getLogger(FractionalHex.class);

    private final double q; // Horizontal coordinate (fractional)
    private final double r; // Diagonal coordinate (fractional)
    private final double s; // Vertical coordinate (calculated as -q - r)

    /**
```

```
     * Validates the fractional cube coordinates (q, r, s).
     * Ensures that q + r + s is approximately equal to 0 (within a small epsilon).
     */
    public FractionalHex {
        if (Math.abs(q + r + s) > 1e-6) { // Allowing small floating-point error tolerance
            logger.error("Invalid FractionalHex coordinates: q + r + s must approximately equal 0. Provided values: q={}, r={}, s={}", q, r, s);
            throw new IllegalArgumentException("Invalid FractionalHex coordinates: q + r + s must approximately equal 0.");
        }
        logger.info("FractionalHex created: {}", this);
    }

    /**
/**
 * Represents the formation of a triangle on a hexagonal grid with anti-aliasing based on 7-point sampling.
 */

public class TriangleFormationWithAntialiasing {

    private Hex vertexA;
    private Hex vertexB;
    private Hex vertexC;
    private List<Hex> triangleHexes;

    public TriangleFormationWithAntialiasing(Hex vertexA, Hex vertexB, Hex vertexC) {
        this.vertexA = vertexA;
        this.vertexB = vertexB;
        this.vertexC = vertexC;
        this.triangleHexes = new ArrayList<>();
        log.info("TriangleFormationWithAntialiasing initialized with vertices: {}, {}, {}", vertexA, vertexB, vertexC);
    }

    public void formTriangle() {
        log.info("Starting triangle formation with vertices: {}, {}, {}", vertexA, vertexB, vertexC);
        triangleHexes.clear();

        List<Hex> edgeAB = generateLine(vertexA, vertexB);
        List<Hex> edgeBC = generateLine(vertexB, vertexC);
        List<Hex> edgeCA = generateLine(vertexC, vertexA);

        fillInterior(edgeAB, edgeBC, edgeCA);

        applyAntiAliasing(edgeAB);
        applyAntiAliasing(edgeBC);
        applyAntiAliasing(edgeCA);

        log.info("Triangle formation completed. Total hexes in triangle: {}", triangleHexes.size());
    }

    private List<Hex> generateLine(Hex start, Hex end) {
        log.debug("Generating line between {} and {}", start, end);
        List<Hex> lineHexes = new ArrayList<>();
        int n = start.distanceTo(end);
        for (int i = 0; i <= n; i++) {
```

```
        Hex interpolated = start.interpolate(end, (double) i / n);
        lineHexes.add(interpolated.round());
    }
    log.debug("Line generated with {} hexes.", lineHexes.size());
    return lineHexes;
}

private void fillInterior(List<Hex> edgeAB, List<Hex> edgeBC, List<Hex> edgeCA) {
    log.info("Filling the interior of the triangle.");
    List<Hex> allEdges = new ArrayList<>();
    allEdges.addAll(edgeAB);
    allEdges.addAll(edgeBC);
    allEdges.addAll(edgeCA);

    for (Hex hex : allEdges) {
        for (Hex neighbor : hex.getNeighbors()) {
            if (!triangleHexes.contains(neighbor)) {
                triangleHexes.add(neighbor);
            }
        }
    }
}

private void applyAntiAliasing(List<Hex> edge) {
    log.info("Applying anti-aliasing to edge with {} hexes.", edge.size());
    for (Hex hex : edge) {
        int coveredPoints = calculateCoveredPoints(hex);
        if (coveredPoints > 0) {
            hex.setOpacity((double) coveredPoints / 7);
            triangleHexes.add(hex);
            log.debug("Anti-aliasing applied to hex: {}, opacity: {}", hex, hex.getOpacity());
        }
    }
}

private int calculateCoveredPoints(Hex hex) {
    int coveredPoints = 0;
    List<Point> samplePoints = hex.getSamplePoints();
    for (Point point : samplePoints) {
        if (isBelowFunction(point)) {
            coveredPoints++;
        }
    }
    log.debug("Hex: {}, covered points: {}/7", hex, coveredPoints);
    return coveredPoints;
}

private Hex center;
private double radius;
private List<Hex> circleHexes;

public AcceleratedCircleFormation(Hex center, double radius) {
    this.center = center;
```

```
        this.radius = radius;
        this.circleHexes = new ArrayList<>();
        log.info("Initialized AcceleratedCircleFormation with center: {}, radius: {}", center, radius);
    }

    public void formCircle() {
        log.info("Starting accelerated circle formation...");
        circleHexes.clear();

        double radiusSquared = radius * radius;

        int approximateRadius = (int) Math.ceil(radius);
        for (int dx = -approximateRadius; dx <= approximateRadius; dx++) {
            for (int dy = -approximateRadius; dy <= approximateRadius; dy++) {
                for (int dz = -approximateRadius; dz <= approximateRadius; dz++) {
                    if (dx + dy + dz == 0) {
                        Hex candidateHex = new Hex(center.getQ() + dx, center.getR() + dy, center.getS() + dz);
                        double distanceSquared = evaluateFunction(candidateHex);

                        if (Math.abs(distanceSquared - radiusSquared) <= 0.5) {
                            circleHexes.add(candidateHex);
                            log.debug("Added hex to circle: {}", candidateHex);
                        }
                    }
                }
            }
        }

        log.info("Circle formation completed. Total hexes in circle: {}", circleHexes.size());
    }

    private double evaluateFunction(Hex hex) {
        int dq = hex.getQ() - center.getQ();
        int dr = hex.getR() - center.getR();
        int ds = hex.getS() - center.getS();

        return dq * dq + dr * dr + ds * ds;
    }

    public int getHexCount() {
        return circleHexes.size();
    }
}

package com.vshmaliukh.hexagon_interpolation;

import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Polygon;
import javafx.stage.Stage;
import lombok.extern.slf4j.Slf4j;
```

```
import java.util.ArrayList;
import java.util.List;

@Slf4j
public class HexagonalRaster extends Application {

    private static final int HEX_RADIUS = 10;

    @Override
    public void start(Stage primaryStage) {
        int radius = 5;
        int centerX = 100;
        int centerY = 100;
        int sectorCount = 6;

        List<List<Polygon>> hexagons = generateHexagonalRaster(radius, centerX, centerY);
        int[] sectorSteps = calculateSectorSteps(radius, centerX, centerY, sectorCount);

        Group root = renderHexagonalRaster(hexagons, centerX, centerY, radius, sectorCount, sectorSteps);

        Scene scene = new Scene(root, 600, 600);
        primaryStage.setTitle("Hexagonal Raster");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    private List<List<Polygon>> generateHexagonalRaster(int radius, int centerX, int centerY) {
        log.info("Generating hexagonal raster with radius {}.", radius);
        List<List<Polygon>> hexagons = new ArrayList<>();

        for (int q = -radius; q <= radius; q++) {
            int r1 = Math.max(-radius, -q - radius);
            int r2 = Math.min(radius, -q + radius);
            List<Polygon> row = new ArrayList<>();

            for (int r = r1; r <= r2; r++) {
                int x = q + centerX;
                int y = r + centerY;
                Polygon hexagon = createHexagon(x, y, HEX_RADIUS);
                row.add(hexagon);
            }
            hexagons.add(row);
        }
        return hexagons;
    }

    private int[] calculateSectorSteps(int radius, int centerX, int centerY, int sectorCount) {
        log.info("Calculating sector steps for the raster.");
        int[] sectorSteps = new int[sectorCount];
        for (int q = -radius; q <= radius; q++) {
            int r1 = Math.max(-radius, -q - radius);
            int r2 = Math.min(radius, -q + radius);
```

```
        for (int r = r1; r <= r2; r++) {
            if (isPointInCircle(q + centerX, r + centerY, radius)) {
                int sector = getSector(centerX, centerY, q, r, sectorCount);
                sectorSteps[sector]++;
            }
        }
    }
    return sectorSteps;
}

private Group renderHexagonalRaster(List<List<Polygon>> hexagons, int centerX, int centerY, int radius, int sectorCount, int[] sectorSteps) {
    log.info("Rendering hexagonal raster.");
    Group root = new Group();
    for (int i = 0; i < hexagons.size(); i++) {
        List<Polygon> row = hexagons.get(i);
        for (int j = 0; j < row.size(); j++) {
            Polygon hexagon = row.get(j);
            int sector = getSector(centerX, centerY, j - radius, i - radius, sectorCount);
            hexagon.setFill(getSectorColor(sector, sectorCount));
            root.getChildren().add(hexagon);
        }
    }
    addSectorLabels(root, centerX, centerY, radius, sectorCount, sectorSteps);
    return root;
}

private Polygon createHexagon(int x, int y, int size) {
    Polygon hexagon = new Polygon();
    for (int i = 0; i < 6; i++) {
        double angle = 2 * Math.PI / 6 * (i + 0.5);
        double newX = x + size * Math.cos(angle);
        double newY = y + size * Math.sin(angle);
        hexagon.getPoints().addAll(newX, newY);
    }
    hexagon.setStroke(Color.BLACK);
    return hexagon;
}

private int getSector(int centerX, int centerY, int x, int y, int sectorCount) {
    double angle = Math.atan2(y - centerY, x - centerX);
    if (angle < 0) angle += 2 * Math.PI;
    return (int) Math.floor(angle / (Math.PI * 2 / sectorCount));
}

private Color getSectorColor(int sector, int sectorCount) {
    double hue = (double) sector / sectorCount * 360;
    return Color.hsb(hue, 1, 1);
}

private boolean isPointInCircle(int x, int y, int radius) {
    int dx = x - radius;
    int dy = y - radius;
    return dx * dx + dy * dy <= radius * radius;
```

```
    }

    private void addSectorLabels(Group group, int centerX, int centerY, int radius, int sectorCount, int[] sectorSteps) {
        log.info("Adding sector labels.");
        double sectorAngle = Math.PI * 2 / sectorCount;
        double labelRadius = radius + 10;

        for (int i = 0; i < sectorCount; i++) {
            double angle = i * sectorAngle;
            double x = centerX + labelRadius * Math.cos(angle);
            double y = centerY + labelRadius * Math.sin(angle);

            javafx.scene.text.Text label = new javafx.scene.text.Text(String.valueOf(sectorSteps[i]));
            label.setX(x);
            label.setY(y);
            group.getChildren().add(label);
        }
    }

    public static void main(String[] args) {
        launch(args);
    }
}
package com.vshmaliukh.hexagon_interpolation;
import lombok.Getter;
import lombok.extern.slf4j.Slf4j;
import java.util.*;
import java.util.stream.Collectors;
/**
 * Class for managing and registering steps based on direction and sector.
 */
@Slf4j
@Getter
public class StepRegister {
    private final int radius;
    private final Set<Step> stepSet = new LinkedHashSet<>();
    public StepRegister(int radius) {
        this.radius = radius;
    }
    /**
     * Adds a step to the register with a given direction and sector.
     * If the step already exists, its counter is incremented.
     *
     * @param stepDirection the direction of the step.
     * @param sector        the sector associated with the step.
     */
    public void addStep(String stepDirection, String sector) {
        stepSet.stream()
                .filter(step -> stepDirection.equals(step.getStepDirection()) && sector.equals(step.getSector()))
                .findFirst()
                .ifPresentOrElse(
                        step -> step.incrementCounter(),
                        () -> stepSet.add(new Step(stepDirection, sector))
```

```
            );
    }
    /**
     * Retrieves the total count of steps for a specific sector.
     *
     * @param sector the sector for which to retrieve the step count.
     * @return the total count of steps in the sector.
     */
    public int getCurrentCountBySector(String sector) {
        return getStepsBySector(sector).stream()
                .mapToInt(Step::getCounter)
                .sum();
    }
    /**
     * Retrieves the total count of steps for a specific step direction and sector.
     *
     * @param stepDirection the direction of the step.
     * @param sector        the sector for which to retrieve the step count.
     * @return the count of steps for the given direction in the sector.
     */
    public int getCurrentCountByStepAndSector(String stepDirection, String sector) {
        return getStepsBySector(sector).stream()
                .filter(step -> stepDirection.equalsIgnoreCase(step.getStepDirection()))
                .mapToInt(Step::getCounter)
                .sum();
    }
    /**
     * Retrieves all steps associated with a specific sector.
     *
     * @param sector the sector for which to retrieve steps.
     * @return a list of steps in the sector.
     */
    public List<Step> getStepsBySector(String sector) {
        return stepSet.stream()
                .filter(step -> sector.equalsIgnoreCase(step.getSector()))
                .collect(Collectors.toList());
    }
    /**
     * Prints the step register's contents in a formatted output.
     */
    public void printResult() {
        log.info("\nR={} |", radius);

log.info("=============================================================================================================
======================");
        List<Step> sortedSteps = stepSet.stream()
                .sorted(Comparator.comparing(Step::getSector).thenComparing(Step::getStepDirection))
                .collect(Collectors.toList());
        sortedSteps.stream()
                .collect(Collectors.groupingBy(Step::getSector))
                .forEach((sector, steps) -> {
                    int totalSteps = steps.stream().mapToInt(Step::getCounter).sum();
                    steps.stream().sorted(Comparator.comparing(Step::getCounter).reversed())
```

```
                .forEach(step -> {
                    double percentage = ((double) step.getCounter() / totalSteps) * 100;
                    log.info("| Sector: {} | Step: {} | Count: {} | Percentage: {:.2f}%",
                            step.getSector(), step.getStepDirection(), step.getCounter(), percentage);
                });
            log.info("-----------------------------------------------------------------------------------------------------------------------");
        });
    }
    /**
     * Inner class representing a step with direction, sector, and counter.
     */
    @Getter
    public static class Step {
        private final String stepDirection;
        private final String sector;
        private int counter = 1;
        public Step(String stepDirection, String sector) {
            this.stepDirection = stepDirection;
            this.sector = sector;
        }
        public void incrementCounter() {
            counter++;
        }
    }
}
package com.vshmaliukh.hexagon_interpolation;
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.LineTo;
import javafx.scene.shape.MoveTo;
import javafx.scene.shape.Path;
import javafx.stage.Stage;
import java.util.*;
public class CanvasDrawing extends Application {
    private static final double TAN_15 = 0.26794919243;
    private static final double TAN_30 = 0.57735026919;
    private static final double TAN_60 = 1.73205080757;
    private static final double TAN_75 = 3.73205080757;
    private static final String SECTOR_0_15 = "0° to 15°";
    private static final String SECTOR_15_30 = "15° to 30°";
    private static final String SECTOR_30_45 = "30° to 45°";
    private static final String SECTOR_45_60 = "45° to 60°";
    private static final String SECTOR_60_75 = "60° to 75°";
    private static final String SECTOR_75_90 = "75° to 90°";
    private static final double HEX_SIZE = 5;
    private static final double WIDTH_COEFFICIENT = Math.sqrt(3) / 2;
    private static final double HEX_WIDTH = HEX_SIZE * WIDTH_COEFFICIENT;
    private static final double HEX_HEIGHT = HEX_SIZE;
    private static final double WIDTH_HALF_DISTANCE = 0.5 * HEX_WIDTH;
    private static final double HEIGHT_ALT_DISTANCE = 0.75 * HEX_HEIGHT;
    private final List<Plot> plots = new ArrayList<>();
```

```
public static void main(String[] args) {
    launch();
}
@Override
public void start(Stage stage) {
    Pane pane = new Pane();
    pane.setStyle("-fx-background-color:white;");
    Path path = new Path();
    path.setStrokeWidth(1);
    path.setStroke(Color.BLACK);
    pane.getChildren().add(path);
    drawCircle(path);
    Scene scene = new Scene(pane, 800, 600);
    scene.setOnScroll(scrollEvent -> {
        double scale = pane.getScaleX() + (scrollEvent.getDeltaY() > 0 ? 0.01 : -0.01);
        pane.setScaleX(scale);
        pane.setScaleY(scale);
    });
    stage.setScene(scene);
    stage.setTitle("Hexagonal Circle Drawing");
    stage.show();
}
private void drawCircle(Path path) {
    double radius = 50;
    double centerX = 0;
    double centerY = 0;
    StepRegister stepRegister = new StepRegister((int) radius);
    double x = radius * HEX_WIDTH;
    double y = 0;
    String previousStep = "";
    String currentStep;
    boolean isSecondStep = false;
    while (x > 0) {
        registerHexagon(path, x, y);
        double OFdL = calculateOF(x - WIDTH_HALF_DISTANCE, y + HEIGHT_ALT_DISTANCE, radius);
        double OFdR = calculateOF(x + WIDTH_HALF_DISTANCE, y + HEIGHT_ALT_DISTANCE, radius);
        if (OFdR < OFdL) {
            x += WIDTH_HALF_DISTANCE;
            y += HEIGHT_ALT_DISTANCE;
            currentStep = "diagonally right";
        } else {
            x -= WIDTH_HALF_DISTANCE;
            y += HEIGHT_ALT_DISTANCE;
            currentStep = "diagonally left";
        }
        if (isSecondStep) {
            stepRegister.addStep(previousStep + " + " + currentStep, SECTOR_0_15);
        } else {
            previousStep = currentStep;
        }
        isSecondStep = !isSecondStep;
    }
    stepRegister.printResult();
```

```
    }
    private double calculateOF(double x, double y, double radius) {
        return Math.abs(x * x + y * y - radius * radius);
    }
    private void registerHexagon(Path path, double x, double y) {
        addHexagonToPath(path, x, y);
        addHexagonToPath(path, -x, y);
        addHexagonToPath(path, x, -y);
        addHexagonToPath(path, -x, -y);
    }
    private void addHexagonToPath(Path path, double x, double y) {
        path.getElements().addAll(
                new MoveTo(x, y),
                new LineTo(x, y - HEX_HEIGHT / 2),
                new LineTo(x + HEX_WIDTH / 2, y - HEX_HEIGHT / 4),
                new LineTo(x + HEX_WIDTH / 2, y + HEX_HEIGHT / 4),
                new LineTo(x, y + HEX_HEIGHT / 2),
                new LineTo(x - HEX_WIDTH / 2, y + HEX_HEIGHT / 4),
                new LineTo(x - HEX_WIDTH / 2, y - HEX_HEIGHT / 4),
                new LineTo(x, y - HEX_HEIGHT / 2)
        );
    }
    private static class Plot {
        private final double x;
        private final double y;
        private Plot(double x, double y) {
            this.x = x;
            this.y = y;
        }
        @Override
        public boolean equals(Object o) {
            if (this == o) return true;
            if (o == null || getClass() != o.getClass()) return false;
            Plot plot = (Plot) o;
            return Double.compare(plot.x, x) == 0 &&
                    Double.compare(plot.y, y) == 0;
        }
        @Override
        public int hashCode() {
            return Objects.hash(x, y);
        }
    }
}
```

Додаток Г (обов'язковий)

Перевірка на плагіат

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ

НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Інформаційна технологія для оптимізації побудови зображень на шестикутному растрі

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра автоматизації та інтелектуальних інформаційних технологій, факультет інтелектуальних інформаційних технологій та автоматизації

Показники звіту подібності Turnitin

Оригінальність	**97**
Схожість	3

Аналіз звіту подібності (відмітити потрібне):

- ■ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
- □ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Роман МАСЛІЙ
(підпис) (прізвище та ім'я)

Ознайомлені з повним звітом подібності, який був згенерований системою Turnitin щодо роботи.

Автор роботи _____ Владислав ШМАЛЮХ
(підпис) (прізвище та ім'я)

Керівник роботи _____ Ілона БОГАЧ
(підпис) (прізвище та ім'я)