КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації інформаційних технологій

Кафедра інформаційних технологій

(назва випускової кафедри)

КВАЛІФІКАЦІЙНА РОБОТА
ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему:

РОЗРОБКА АВТОМАТИЗОВАНОЇ ВЕБ-СИСТЕМИ ПЕРЕТВОРЕННЯ ЗОБРАЖЕНЬ У ПІКСЕЛЬ-АРТ

Суткова Анастасія Олександрівна

Київ 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет автоматизації інформаційних технологій

Кафедра інформаційних технологій

ЗАТВЕРДЖУЮ
Завідувач кафедри ІТ
Тетяна Гончаренко
_____
„___”_____2025 року

КВАЛІФІКАЦІЙНА РОБОТА ЗДОБУВАЧА СТУПЕНЯ ВИЩОЇ ОСВІТИ БАКАЛАВР

на тему: «**Розробка автоматизованої веб-системи перетворення зображень у піксель-арт**»

Я як здобувач вищої освіти КНУБА розумію і підтримую політику закладу з академічної доброчесності. Я не надавав(-ла) і не одержував(-ла) недозволену допомогу під час підготовки цієї роботи. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Здобувач
Суткова Анастасія Олександрівна
122 «Комп’ютерні науки»
(спеціальність)
Інформаційні управляючі системи і технології
(освітня програма)
Групи КН-21-2
Керівник Рябчун Ю.В.
(прізвище та ініціали)
Доктор філософії
(вчене звання, науковий ступінь)
Рецензент к.т.н., доц. Саченко І.А.
(Прізвище та ініціали)

Ідентичність підтверджую

Київ, 2025 р.

КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БУДІВНИЦТВА І АРХІТЕКТУРИ

Факультет: Автоматизації інформаційних технологій

Випускова кафедра: Інформаційних технологій

Освітній ступінь: Бакалавр

Спеціальність: Комп'ютерні науки

Освітня програма: Інформаційні управляючі системи та технології

ЗАТВЕРДЖУЮ

Завідувач кафедри ІТ

Тетяна ГОНЧАРЕНКО

_____

„___"_____2025 року

З А В Д А Н Н Я

ДО ВИКОНАННЯ КВАЛІФІКАЦІЙНОЇ ВИПУСКНОЇ РОБОТИ НА ЗДОБУТТЯ ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВР

Суткова Анастасія Олександрівна

1. Тема роботи РОЗРОБКА АВТОМАТИЗОВАНОЇ ВЕБ-СИСТЕМИ ДЛЯ ПЕРЕТВОРЕННЯ ЗОБРАЖЕНЬ У ПІКСЕЛЬ-АРТ
затверджена наказом ректора КНУБА № 235/23/25 від «14» лютого 2025 року
2. Керівник роботи Рябчун Юлія Володимирівна, PhD
(прізвище, ім'я та по батькові, науковий ступінь, вчене звання)
3. Строк подання Здобувачем роботи до захисту _____
4. Зміст пояснювальної записки за розділами:
Р.1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ
Р.2 МЕТОДОЛОГІЯ ТА ВИБІР АЛГОРИТМІВ
Р.3 ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ ВЕБ-СИСТЕМИ
Р.4 ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ

5. Графічний матеріал за розділами:
Р.1 7 рисунків, 7 таблиць
Р.2 22 рисунки
Р.3 5 таблиць, 8 рисунків
Р.4 3 таблиці

6. Календарний план виконання роботи:

Види робіт та їх зміст	Дата виконання
Розділ 1	20.01.2025
Розділ 2	15.02.2025
Розділ 3	22.04.2025
Розділ 4	22.04.2025
Остаточне оформлення роботи	24.05.2025
Направлення роботи для перевірки на плагіат	24.05.2025
Попередній захист роботи на випусковій кафедрі	26.05.2025
Направлення роботи на рецензування	26.05.2025

7. Консультанти розділів атестаційної випускної роботи

Розділ	Прізвище, ініціали та посада консультанта	Перевірив	
		дата	підпис
Розділ 1	Рябчун Ю.В., доц.каф.ІТ	20.01.2025	
Розділ 2	Рябчун Ю.В., доц.каф.ІТ	15.02.2025	
Розділ 3	Рябчун Ю.В., доц.каф.ІТ	22.02.2025	
Розділ 4	Рябчун Ю.В., доц.каф.ІТ	22.04.2025	

8. Дата видачі завдання _____

Зав. кафедри _____ _____
(підпис) (прізвище та ініціали)
Керівник _____ _____
(підпис) (прізвище та ініціали)
Здобувач _____ _____
(підпис) (прізвище та ініціали)

РЕЗЮМЕ (SUMMARY) *до кваліфікаційної випускної роботи* *Здобувача:*	Суткова Анастасія Олександрівна Sutkova Anastasia		
ЗВО	Київський національний університет будівництва і архітектури		
Тема (українською та англійською)	Розробка автоматизованої веб-системи перетворення зображень у піксель-арт		
Освітній ступінь	Бакалавр		
Факультет	Автоматизації і інформаційних технологій		
Випускова кафедра	Інформаційних технологій		
Спеціальність	122 «Комп’ютерні науки»		
Освітня програма	Інформаційні управляючі системи та технології		
Керівник	Рябчун Юлія Володимирівна		
Обсяг роботи:	пояснювальна записка, стор.	розділів	Кількість слайдів презентації
	115	4	
Розділ 1.	АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ		
Розділ 2.	МЕТОДОЛОГІЯ ТА ВИБІР АЛГОРИТМІВ		
Розділ 3.	ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ ВЕБ-СИСТЕМИ		
Розділ 4.	ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ		

Ключові слова:	автоматизована веб-система, піксель-арт, перетворення зображень, масштабування зображень, кластеризація кольорів
Keywords:	automated web system, pixel art, image conversion, image scaling, color clustering

АНОТАЦІЯ

Суткова А.О. Розробка автоматизованої веб-системи для перетворення зображень у піксель-арт.

Атестаційна випускна робота бакалавра за спеціальністю 122 «Комп'ютерні науки», освітня програма «Інформаційні управляючі системи та технології». - Київський національний університет будівництва та архітектури. - Київ, 2025.

Дана дипломна робота присвячена розробці веб-системи, яка автоматизує процес перетворення фотографій або ескізів у стиль піксель графіки. У роботі реалізовано функціонал системи, включаючи завантаження зображень, їх обробку з використанням алгоритмів зменшення роздільної здатності та кластеризації кольорів, а також можливість ручного налаштування результату. Практична частина містить інтеграцію сучасних інструментів обробки зображень та адаптивних фільтрів, що забезпечують високу якість перетворення. Проведено тестування системи для оцінки точності роботи алгоритмів, продуктивності та зручності використання веб-інтерфейсу. На основі аналізу результатів розробки сформульовано висновки щодо ефективності впроваджених рішень, а також надаються рекомендації для подальшого вдосконалення системи та користувацького досвіду.

Робота викладена на: 123 аркушах, містить 2 додатки, 15 таблиць, 37 рисунків, список використаної літератури із 36 найменувань.

Ключові слова: автоматизована веб-система, піксель-арт, перетворення зображень, масштабування зображень, кластеризація кольорів.

SUMMARY

Sutkova A. O. Development of an automated web system for converting pictures into pixel art style. Bachelor's thesis in the specialty: 122 "Computer Science," specialization: "Information Management Systems and Technologies." - Kyiv National University of Construction and Architecture. - Kyiv, 2025.

This thesis is devoted to the development of a web system that automates the process of converting pictures into pixel art style.

The work implements the functionality of the system, including loading images, processing them using algorithms for reducing resolution and color clustering, as well as the ability to manually adjust the result. The practical part includes the integration of modern image processing tools and the use of adaptive filters to ensure high quality conversion. The system was tested to assess the accuracy of the algorithms, performance, and ease of use of the web interface. Based on the development results, conclusions were formulated regarding the effectiveness of the implemented solution, as well as recommendations for further improvement of the system and user experience.

The work is presented on: 122 sheets, contains 2 appendices, 15 tables, 37 figures, a list of used literature of 36 titles.

Keywords: automated web system, pixel art, image conversion, image scaling, color clustering.

ЗМІСТ

ВСТУП

У сучасному світі цифрових технологій піксельний стиль займає доволі значну нішу в дизайні та активно використовується в різних сферах застосування 2D-графіки, зокрема у відеоіграх, дизайні інтерфейсів і різних видів анімацій. Його популярність пояснюється естетикою мінімалізму, яка дозволяє створювати візуально привабливі елементи з мінімальними ресурсами, а також: ностальгічним стилем ретро-графіки, що відсилає до ранньої епохи відеоігор. Окрім того, піксель-арт є ефективним засобом оптимізації ресурсів завдяки своїй простоті виконання - це робить його ідеальним рішенням для мобільних додатків та веб-систем. Однак, створення такого стилю за допомогою роботи художників досі залишається досить складним і трудомістким завданням, що вимагає багато людських зусиль та навичок, а часто і дорогоцінного часу між щільними графіками проекту. Його вибагливість створює труднощі для дизайнерів-початківців, які не мають досвіду у піксельній графіці, та значно обмежує можливість масового застосування у сучасному цифровому контенті.

Сьогодні автоматизація процесу перетворення готових зображень у піксельний малюнок є затребуваним інструментом адже графічний редактор, який автоматично конвертує фотографії або ескізи у стиль пікселю, зможе значно спростити процес малювання як для художників-професіоналів, так і відкрити нові можливості для художників-початківців. І хоча автоматизація є лише частиною конвеєра створення візуального контенту - це саме той випадок, коли алгоритми і автори цього контенту можуть взаємодіяти з зображеннями у спільному процесі. Така система і підхід збільшать інструментарій та нададуть змоги створювати графіку високої якості за короткий проміжок часу.

Метою даної роботи є розробка веб-орієнтованої автоматизованої системи для перетворення зображень у піксельну графіку. Передбачається створення ефективного та зручного інструменту, який може обробляти та редагувати зображення, а також: адаптувати їх до заданих користувачем параметрів стилю піксельного мистецтва. Система повинна враховувати вимоги до якості, точності і

швидкості обробки, а інструмент має бути доступним для широкого кола користувачів.

Обґрунтування проектного рішення:

Для реалізації поставлених цілей в даній дипломній роботі буде створений веб-застосунок з впровадженням сучасних алгоритмів обробки зображень, а саме: алгоритми зменшення роздільної здатності, метод кластеризації кольорів і адитивні фільтри для різноманітних стилізацій зображення.

Можливі галузі застосування:

Вихідні зображення що є результатами роботи автоматизованого редактора зображень можуть широко використовуватися у різних галузях що застосовують 2D-графіку, ось наприклад:

- Відеоігри (ретро-графіка для платформерів, RPG та інших ігрових жанрів, де естетика піксель-арту є ключовою);
- Дизайн інтерфейсів (піксель використовується для створення унікальних впізнаваних інтерфейсів);
- Реклама та маркетинг (використання піксель-арту у візуальних матеріалах сприяє впізнаванності продукту, що рекламується);

Об'єкт дослідження - процес перетворення зображень у чіткий піксель-арт.

Предмет дослідження - алгоритми обробки зображень, такі як кластеризація кольорів та методи зменшення роздільної здатності.

Основні методи дослідження: аналіз наукової літератури та компонентів реалізації веб-додатку, тестування системи та техніко-економічне обґрунтування розробки.

Загалом це дослідження сприятиме технологічному розвитку у галузі 2D-графіки, розширюючи можливості для автоматизації творчого процесу та знижуючи поріг входження для користувачів з різним досвідом роботи з піксельним малюнком.

Проект також важливий тим, що має потенціал сприяти створенню цифрових продуктів, які підвищать креативність і продуктивність українських геймдизайнерів та розробників. Також цілком можливо, що реалізація цього та подібних інструментів сприятиме зростанню конкурентоспроможності українських цифрових продуктів на глобальному світовому ринку.

Розділ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні характеристики об'єкту дослідження

Піксельна графіка або Pixel Art - це особливий вид цифрового зображення, що створюється за допомогою растрового графічного редактора та редагується на рівні найдрібніших одиниць цифрового зображення - пікселів. В результаті роздільна здатність фотографії або ескізу настільки мала, що чітко видно окремі пікселі, а весь малюнок ніби складається лише з блоків. На старих комп'ютерах, ігрових приставках і багатьох мобільних телефонів переважно використовується саме піксельна графіка оскільки вона є єдиним способом зробити чітким невелике зображення при малій роздільній здатності екранів цих пристроїв. [1]

Загалом, невеликі розміри піксельного малюнку і є особливістю, що відрізняє його від інших видів цифрового мистецтва. Також йому притаманна обмежена кольорова палітрою і, як правило, відсутнє будь-яке згладжування. В інструментарії піксельної графіки лише найпростіші інструменти, а саме: олівець, гумка та заливка.

Історія та розвиток піксель-арту

Термін «pixel art» вперше використала Аделла Ґолдберт і Роберт Флеґал з дослідницького центру Пало-Альто корпорації «Xerox» ще в 1982 році. Проте сама піксельна графіка використовувалася ще за багато років до цього, коли апаратні обмеження комп’ютерів і ігрових консолей стали причиною використання низької роздільної здатності та малих кольорових палітр. Хронологія років і як змінювалося використання цього стилю представлені у таблиці 1.1.

Отже, стиль пікселю набув свого широкого розповсюдження в 1980-ті роки на ПК та ігрових консолях, але з появою 256-кольорових палітр був витіснений з робочих столів операційних систем. Незважаючи на це, на портативних пристроях, таких як: мобільні телефони, PSP і Nintendo DS - піксельний малюнок залишається широко поширеним і сьогодні.

Наразі піксель-арт надає широкого зсуву у сприйнятті цифрової та ностальгічної естетики. Сучасний піксель-арт є дещо відповідною реакцією любителів минулих ігор та малюнків на нинішнє переважання тривимірної графіки у

просторах цифрового ринку. Серед художників він є популярним як динамічна форма мистецтва, що має потенціал розвитку в майбутньому.

Таблиця 1.1

Ключові періоди в історії піксель-арту

Рік	Опис
1950-ті	З’являється перша комп’ютерна графіка, що відкриває шлях до цифрових дисплеїв на основі пікселів.
1972	Рік створення Space Invaders - однієї з перших відеоігор у піксельній графіці.
1980-ті	Ера домашніх 8-бітних комп’ютерів та ігрових консолей з обмеженими графічними можливостями (Atari, Commodore 64 і т.д).
1983	Рік першого в історії буму відеоігор, спричиненого виходом на ринок ігрової консолі Nintendo Entertainment System (NES).
1989	Відбувся 16-бітний прогрес, Sega Genesis і Super Nintendo Entertainment System (SNES) підняли піксель на нові висоти.
1990-ті	Золотий вік піксельного мистецтва у класичних іграх, таких як Sonic the Hedgehog, Final Fantasy VI і т.д.
1996	Через перехід на 3D-графіку, попит на піксель в іграх зменшується.
2000-ті	Поява інді-ігор від незалежних ігрових студій повертає піксель-арт в центр уваги (наприклад Braid та Fez).
2010-ті	Роки відродження популярності завдяки таким іграм, як Stardew Valley, Celeste та іншим наддеталізованим представникам інді-ігор у стилі піксель-арту.
2020-ті	Піксельне графіка продовжує бути на слуху в мейнстрімових медіа, зокрема в спільнотах реклами, анімації та цифрового мистецтва. Соціальні платформи перетворилися на віртуальні онлайн-галереї, які дозволяють художникам демонструвати свої роботи абсолютно різній світовій мистецькій аудиторії.

Піксель-арт як форма графіки

Піксель-арт характеризується використанням низької роздільної здатності та обмеженої палітри кольорів. Завдяки цим двом правилам художники створюють роботи, які поєднують ретро-естетику та сучасні візуальні ефекти.

На відміну від векторного стилю, який здебільшого базується на математичних формулах, у піксель-арті кожен піксель використовується як окремий елемент, що створює сітчасту точність зображень. Це робить стиль унікальним, адже в його навмисні обмеження стають джерелом творчого вираження (рис 1.1).

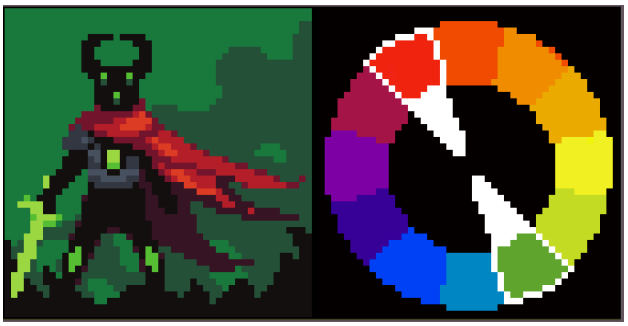


Рис. 1.1. Приклад обмеженої кольорової палітри піксельного малюнку

Переваги та недоліки піксельної графіки представлені у таблиці 1.2.

Таблиця 1.2

Переваги та недоліки піксель-арту

Переваги	Недоліки
⇒ Піксель-арт є одним з найбільш простих у вивченні стилів комп'ютерного мистецтва. Простий піксельний малюнок можливо намалювати навіть не маючи особливих художніх здібностей та навичок;	⇒ В епоху hicolor-моніторів і відеопроцесорів з апаратним альфа-змішуванням більш виразно виглядають інші стилі. Проте на низьких дозволах все одно відбувається вирівнювання ліній по пікселям;
⇒ Він легкий у створенні, адже є невибагливим до програмного забезпечення. Створити малюнок можливо за допомогою майже будь-якого з базових графічних редакторів. Для розробників ігор немає потреби витрачати тижні чи місяці, як на створення складних	⇒ Погано переносить автоматичне масштабування. При зміненні формату розширення екрану зображення необхідно перемальовувати. І хоча на сучасних ПК розширення моніторів достатньо високе для масштабування пікселю - при

3D-моделей або розширених текстур, достатньо лиш творчо і повторно використовувати ресурси.	обмежених параметрах екранів залишається запускати ті ж ігри лише у форматі вікон;
⇒ Обмеження змушують художників бути більш винахідливими, що зрештою призводить до більш елегантних і ефективних рішень;	⇒ На неякісних моніторах, «сітчасте тонування» може мерехтіти, що у свою чергу є неприємним подразником для людського ока;
⇒ За рахунок застосування палітрових форматів з обмеженою кількістю кольорів та малої роздільної здатності, зображення у цьому стилі є легкими для внутрішньої пам’яті пристроїв;	-
⇒ Навіть за умов дуже поганої передачі кольору піксельний малюнок не втрачає виразності та добре виглядає на екранах із чіткими межами пікселів;	-

Підсумовуючи, піксель-арт вирізняється серед усіх стилів своєю простотою освоєння, мінімальними вимогами до програмного забезпечення та компактністю зображень. Водночас обмежена кольорова палітра й фіксована роздільна здатність ускладнюють автоматичне масштабування і можуть призводити до артефактів на сучасних дисплеях.

Пікселізація

Пікселізація - це навмисне спрощення деталей зображення або його частин задля приховання деякого змісту. Одним з найпоширеніших застосувань пікселізації є захист конфіденційності, а також дотримання стандартів медіа, з метою цензурування забороненого змісту фотографій або відео(рис 1.2). Пікселізація є одним з найшвидших і найефективніших інструментів для розмивання облич у натовпі, задля збереження анонімності з міркувань безпеки, або для приховання наготи чи вульгарних жестів, що можуть бути неприємними для вразливої аудиторії. Ця техніка дозволяє глядачеві зрозуміти контекст зображень, не розкриваючи подробиць, які можуть бути приватними, образливими чи відволікаючими. [2]

Рис. 1.2. Приклад зображення з пікселізацією обличчя

Пікселізація також може бути використана для спрямування фокусу у зображенні. Розмиваючи пікселем певні ділянки, увагу глядача можна спрямувати на чисті зони (рис 1.3). Ця техніка особливо корисна для освітнього контенту або демонстрації продуктів, коли потрібно виділити певні особливості, не відволікаючи увагу на інший вміст.

Рис. 1.3. Приклад спрямованого фокусу зображення

1.2 Огляд і аналіз існуючих рішень

Сьогодні піксель-арт переживає справжній ренесанс, адже завдяки своїй доступності та мінімалізму він ідеально підходить для різних сучасних платформ, мобільних додатків та інді-ігор. Ручне виконання хоча й дарує безліч можливостей для творчого самовираження, все ж залишається складним процесом який вимагає від художника витримки, навичок і часу. Тому, задля спрощення етапів створення піксель-арту на цифровому ринку з’явилося багато спеціалізованих програм які полегшують роботу над малюнком та адаптують його під різні потреби користувачів. Одним з найбільш відомих інструментів у світі піксель-арту є Aseprite. Цей програмний продукт став стандартом серед розробників інді-ігор та художників завдяки своїй функціональності та зручності.

1.2.1 Aseprite

Aseprite - це спеціалізований пропрієтарний редактор зображень, доступний із вихідного коду, розроблений переважно для малювання та анімації піксельного мистецтва. Він працює в середовищах Windows, macOS і Linux, та містить різні інструменти для редагування зображень і анімації. Деякі з його функцій включають:

- Шари та кадри з групуванням шарів;
- Трансформації та інструменти, специфічні для піксельної графіки (pixel-perfect режими, користувацькі пензлі, тощо);
- Попередній перегляд анімації;
- Налаштування кольорової палітри, включаючи 65 палітр за замовчуванням;
- Кольорові профілі та режими (RGBA, індексований та у градаціях сірого);

Вперше випущений у 2010-х роках, цей редактор швидко здобув популярність серед художників і розробників, особливо серед тих, хто працює над проектами у стилі ретро. Програма містить інтуїтивний інтерфейс (рис. 1.4) із розширеним функціоналом, що дозволяє вирішувати як прості, так і складні завдання у сфері 2D-графіки.[3]

Рис. 1.4. Інтерфейс Aseprite

Основні можливості Aseprite:

Однією з головних переваг Aseprite є його орієнтованість на піксельну графіку. Це означає, що всі інструменти програми розроблені з урахуванням специфіки роботи з пікселями. Також Aseprite дозволяє користувачам створювати й зберігати власні палітри кольорів, що є ключовим для художників, які працюють у певному витриманому стилі. Наприклад, користувач може створити свою обмежену кольорову палітру, що характерна для ретро-ігор, або адаптувати палітру під більш сучасні проекти.

Ще однією з важливих особливостей Aseprite є підтримка багатьох форматів файлів, включаючи .ase (власний формат програми), .png, .gif та .jpg. Власний формат зберігає всі дані про шари та анімації, завдяки чому користувачеві легко редагувати проект у майбутньому якщо він захоче до нього повернутися. Завдяки підтримці високої роздільної здатності та можливості працювати з піксельною точністю, Aseprite є ідеальним інструментом для художників, які створюють складні сцени або динамічних персонажів.

Також великою перевагою є забезпечення програмою сумісності із популярними ігровими рушіями, а саме: Unity та Unreal Engine. Тому Aseprite є вибором номер один серед інді-розробників, оскільки він в одному застосунку забезпечує всім необхідним для створення ігрової графіки.

Важливо також відмітити що Aseprite має активну спільноту користувачів які часто діляться своїм досвідом, створюють навчальні матеріали та відповідають на

запитання новачків. Така особливість робить цей редактор не лише інструментом, а й частиною суспільної екосистеми, що поширює та підтримує розвиток піксельного мистецтва.

Переваги та недоліки Aseprite представлені у таблиці 1.3.

Таблиця 1.3

Переваги та недоліки Aseprite

Переваги	Недоліки
⇒ Інтуїтивний інтерфейс робить цей редактор популярним серед новачків і професіоналів;	⇒ На відміну від деяких альтернатив, програма є платною;
⇒ Програма підтримує роботу з палітрами кольорів, дозволяючи зберігати власні підбірки;	⇒ Через відсутність повноцінної роботи з фільтрами автоматизована оптимізація графіки неможлива;
⇒ Експорт у популярних форматах, наприклад, .gif та .png, дозволяє легко інтегрувати графіку в ігрових рушіях;	⇒ Немає вбудованої автоматизації для перетворення зображень у піксель-арт. Цей інструмент розроблений лише для ручного створення графіки та анімації;

Отже, Aseprite дозволяє працювати з шарами, кольоровими палітрами та іншими потужними функціями, процес конверсії фотографій у піксель-арт залишається ручним. Наприклад, користувачеві потрібно самостійно знижувати роздільну здатність зображення та вручну налаштовувати деталі та кольори до стилістики піксель-арту, аби досягти бажаного результату.

Таким чином, Aseprite більше підходить для художників, які працюють над створенням графіки "з нуля" або анімацій, ніж для автоматичного перетворення складних зображень у піксель-арт. Це обмеження робить його менш універсальним для тих, хто шукає швидкі способи автоматизації процесу, але водночас надає величезні можливості для художнього контролю і творчості.

1.2.2 Piskel

Наступним інструментом є Piskel - це безкоштовний графічний редактор з відкритим вихідним кодом, розроблений для створення піксельних зображень та покадрової анімації. Його простий інтерфейс (рис. 1.5) і доступність роблять його ідеальним вибором як для новачків, так і для професійних художників, які працюють із піксельною графікою. Особливістю Piskel є те, що він побудований виключно на JavaScript, HTML та CSS та доступний у двох формах: як веб-додаток, який працює прямо у браузері, і як десктопна програма для Windows, macOS і Linux. [4]

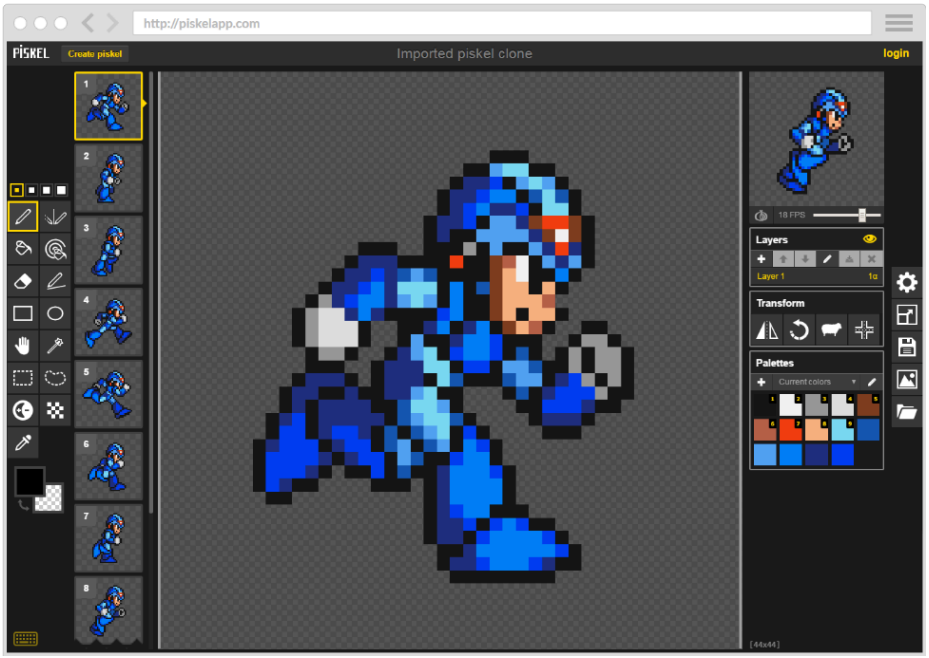


Рис. 1.5. Інтерфейс Piskel

Основні можливості Piskel:

Piskel має інтуїтивно зрозумілий інтерфейс, який дозволяє швидко почати роботу без довгого навчання та підготовки. Основні інструменти, такі як: олівець, гумка, інструменти заливки та вибору кольорів - доступні на панелі управління й відразу готові до використання.

Веб-версія Piskel працює швидко навіть на пристроях із невеликими ресурсами, що робить цей редактор доступним для широкої аудиторії. Користувачам не потрібно завантажувати чи встановлювати програму - достатньо лише відкрити браузер і

почати в ньому працювати. Крім того, всі створені роботи можуть зберігатися як локально, так і в хмарному середовищі на сервері Piskel.

Також важливою перевагою Piskel є те, що він є абсолютно безкоштовним і має відкритий вихідний код. Просунуті користувачі можуть як завгодно модифікувати програму, додаючи функції або адаптуючи її під свої власні потреби.

На відміну від Aseprite, який пропонує більше професійних функцій для роботи з шарами та анімацією, Piskel орієнтований на простоту й доступність. Його головною метою є забезпечення швидкого й зручного створення піксельного малюнку без зайвих ускладнень.

Переваги та недоліки Piskel представлені у таблиці 1.4.

Таблиця 1.4

Переваги та недоліки Piskel

Переваги	Недоліки
⇒ Редактор є безкоштовним і має відкритий вихідний код;	⇒ У порівнянні з тим ж Aseprite, Piskel має менше можливостей для роботи зі складною графікою;
⇒ Має досить простий і зрозумілий інтерфейс;	⇒ Простий інтерфейс є зручним для новачків, проте художники-професіонали можуть відчувати брак функціоналу;
⇒ Дозволяє працювати у веб-застосунку без потреби в інсталяції програмного забезпечення;	-

Отже, Piskel як і Aseprite не має повноцінної функції автоматичного перетворення фотографій або складних зображень у піксель-арт, і може так само запропонувати лише самостійну обробку забраження з попередньо навмисно зменшеною роздільною здатністю.

1.2.3 Grafx2

Grafx2 - це редактор, орієнтований на створення піксель-арту та графіки у стилі ретро (рис. 1.6). Він розроблений як сучасний аналог програм 90-х років, таких як Deluxe Paint та Brilliance, і пропонує багатий набір інструментів, які чудово підходять для роботи з обмеженими палітрами кольорів та створення графіки, характерної для старих ігрових платформ, як-от Amiga, Commodore 64 і DOS. [5]

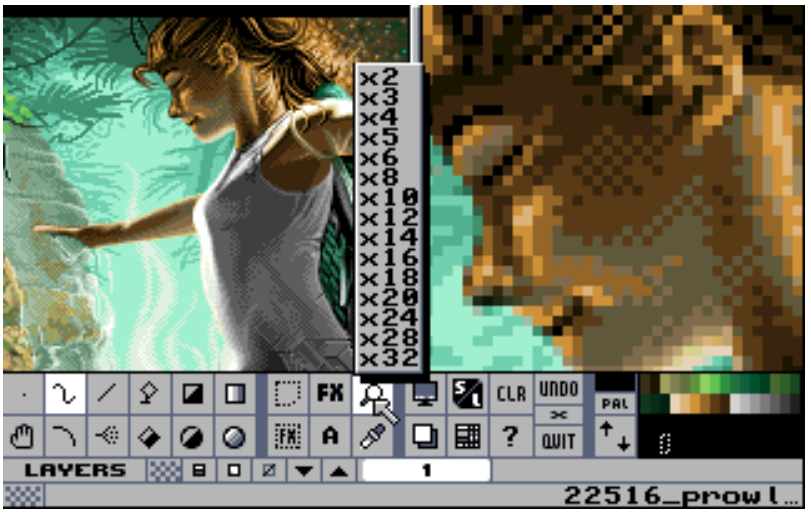


Рис. 1.6. Інтерфейс Grafx2

Основні можливості Grafx2:

Grafx2 створений для тих, хто працює із стилями графіки 8-бітних і 16-бітних епох, адже він ідеально підходить для художників, які хочуть створювати піксельні зображення з давнім автентичним виглядом, характерним для ранньої епохи відеоігор. У програмі є вбудована підтримка обмежених кольорових палітр, що дозволяє створювати графіку, яка відповідає технічним обмеженням ретро-платформ.

Однією з головних переваг Grafx2 є гнучка робота з кольоровими палітрами. Програма дозволяє створювати, редагувати та зберігати власні палітри, а також використовувати попередньо задані набори кольорів, наприклад, для платформ ZX Spectrum або Game Boy.

Також Grafx2 пропонує кілька функцій, які спрощують процес створення графіки, а саме: автоматичне генерування переходів між кольорами або ж функція

створення текстур, яка дозволяє зменшити обсяг ручної роботи при генерації повторюваних візерунків.

Переваги та недоліки Grafx2 представлені у таблиці 1.5.

Таблиця 1.5

Переваги та недоліки Grafx2

Переваги	Недоліки
⇒ Підтримка ретро-форматів, наприклад, Amiga IFF, робить Grafx2 зручним для створення графіки у старих стилях;	⇒ Grafx2 не надає повного функціоналу для створення складних зображень;
⇒ Користувач може налаштувати інтерфейс як йому зручно;	⇒ Інтерфейс може бути складним для освоєння новачками, які ще не знайомі з ретро-стилями роботи;
⇒ Відкритий код надає можливість розширювати функціонал інструменту;	-

Хоч Grafx2 і пропонує спрощений автоматизований функціонал, однак для створення піксель-арту з фотографій користувачеві все ще потрібно виконувати більшість процесів вручну, а саме: знижувати роздільну здатність, адаптувати палітри кольорів і виправляти деталі.

1.2.4 Adobe Photoshop

Adobe Photoshop - це один з найбільш популярних і універсальних інструментів для роботи з графікою, який вже багато років залишається лідером у сфері цифрового графічного мистецтва. Хоча Photoshop не є спеціалізованим редактором для піксель-арту, його багатий функціонал і гнучкість дозволяють адаптувати його й для створення піксельної графіки (рис. 1.7).

Завдяки наявності точних інструментів для роботи з пікселями, підтримці шарів та широкому вибору налаштувань, Photoshop став важливим інструментом для тих, хто хоче створювати піксельний стиль у сучасному цифровому ландшафті. [6]

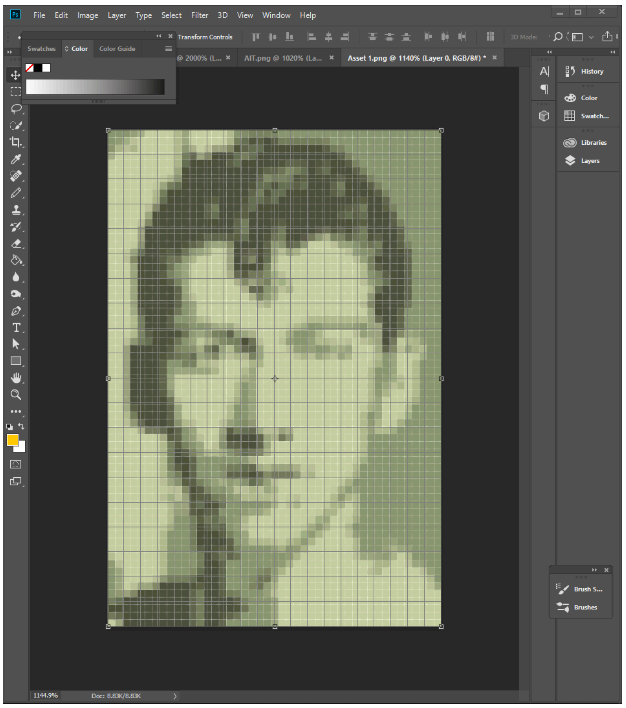


Рис. 1.7. Інтерфейс Photoshop

***Основні можливості Photoshop*:**

Photoshop дозволяє працювати на рівні кожного пікселя завдяки широкому асортименту графічних інструментів та пензликів. Крім того, у Photoshop є функція Pixel Grid, яка активується під час зменшення масштабу полотна і показує чітку сітку пікселів, яка дозволяє художникам бачити кожен елемент у деталях і уникати помилок під час малювання.

Крім цього, Photoshop дає змогу точно керувати кольоровими палітрами завдяки режиму Indexed Color та редактору Color Table. Художник може сформувати власну обмежену палітру, зберегти її окремим файлом, а потім швидко застосовувати до нових зображень або обмінюватися нею з іншими учасниками проєкту. Для згладжування переходів між тонами доступні алгоритми дифузійного дитерингу, що допомагають імітувати плавні градації без збільшення кількості кольорів.

Photoshop також надає інструменти для масштабування зображень без втрати якості. Для піксель-арту особливо важливий режим Nearest Neighbor під час масштабування, який зберігає чіткі краї пікселів, запобігаючи їх розмиттю. Це

дозволяє збільшувати або зменшувати зображення, не порушуючи його піксельної структури.

Переваги та недоліки Photoshop у таблиці 1.6.

Таблиця 1.6

Переваги та недоліки Photoshop

Переваги	Недоліки
⇒ Інструмент надає багатий набір функцій для точної роботи з графікою;	⇒ Програма не є безкоштовною, створює фінансовий бар'єр для користувачів;
⇒ Photoshop має одну з найбільших спільнот користувачів, яка надає доступ до численних навчальних ресурсів, уроків і форумів;	⇒ Photoshop через свій досить широкий функціонал є складним інструментом для освоєння.
⇒ Photoshop підтримує безліч плагінів, які розширюють його функціонал. Деякі з них також спеціалізуються на роботі з піксель-артом та автоматизацією процесів;	⇒ Програма потребує потужного апаратного забезпечення й великого обсягу оперативної пам'яті, тому на малопродуктивних або застарілих комп'ютерах програма може працювати повільно або нестабільно;

Photoshop, як і попередні редактори, не має спеціалізованого інструменту для автоматизованого перетворення зображень у піксель-арт. Він містить функціонал фільтрів, які дозволяють частково автоматизувати процес, а саме: Pixelate (Мозаїка), Threshold (Поріг) та Posterize (Постеризація) - проте вони не надають повністю автоматизований процес перетворення зображення в піксель-арт.

1.2.5 Порівняння існуючих рішень

Отже, розглянуті редактори для створення піксельного мистецтва демонструють різні підходи до роботи з графікою, але кожен з них має як переваги, так і обмеження. Наочне порівняння редакторів представлено в таблиці 1.7;

Таблиця 1.7

Порівняння розглянутих редакторів

Інструмент	Переваги	Недоліки
Aseprite	Інтуїтивний інтерфейс, розширені можливості для створення графіки "з нуля", підтримка популярних форматів файлів, орієнтований на піксель-арт.	Платний, відсутня автоматизація перетворення зображень у піксель.
Piskel	Безкоштовний, простий у використанні, наявність веб-застосунку дозволяє працювати без інсталяції.	Малий функціонал, орієнтований більше на новачків.
Grafx2	Орієнтований на ретро-стиль, підтримує обмежені кольорові палітри, гнучкий у налаштуваннях.	Складний для новачків.
Photoshop	Універсальний, підтримує фільтри для спрощення роботи з пікселями.	Складний у освоєнні, платний.

Загалом Aseprite пропонує потужні можливості для ручного створення піксельного мистецтва, але не має автоматизованого рішення для перетворення фотографій у піксель-арт. Piskel є безкоштовним інструментом що містить в собі базову функціональність для початківців, проте обмежений у функціоналі для більш професійних художників. Водночас така універсальна програма, як Photoshop, пропонує часткову автоматизацію процесу обробки зображень за допомогою фільтрів, проте залишається складними, громіздким та дорогим у використанні.

Кожен графічний редактор має свої переваги й обмеження. Вибір залежить від конкретних завдань користувача та його рівня художніх навичок. Однак, більшість з них не пропонує автоматизованих механізмів для перетворення зображень у піксель-арт, що створює потребу в нових рішеннях, таких як розробка спеціалізованої веб-системи для автоматизації цього процесу для його полегшення та скорочення часових витрат.

1.3 Проблеми та вимоги

У сучасному світі, де візуальна комунікація відіграє важливу роль а створення піксель-арту набуває дедалі більшої популярності, особливо у сфері 2D-графіки та ігрової індустрії. Однак процес перетворення зображень у стиль піксель-арту стикається із численними проблемами та викликами.

Основною проблемою є складність адаптації алгоритмів до різних типів зображень. Сучасні алгоритми зменшення роздільної здатності часто недостатньо враховують унікальні особливості кожного зображення, що призводить до втрати деталей або зниження якості. Зокрема, лінії можуть розмиватися, а ключові деталі - спотворюватися.

Ще однією проблемою є низький рівень автоматизації в уже існуючих рішеннях. Такі сучасні графічні редактори як Aseprite, Piskel, Grafx2 часто вимагають значної ручної роботи, яка уповільнює процес і обмежує продуктивність. Відсутність інтерактивності, такої як: підтримка кольорових палітр або налаштування гарячих клавіш - також зменшує зручність використання.

Значною технічною перешкодою ще є недостатня підтримка сучасних форматів файлів. Крім того, існують проблеми з масштабуванням зображень з втратою якості самого зображення. Відсутність адаптації алгоритмів для різноманітних сценаріїв та типів зображень значно звужує можливості використання існуючих інструментів, створюючи потребу у вдосконалених, універсальних рішеннях.

Рекомендації для веб-додатків:

- Кросплатформеність - веб-додаток має коректно працювати на різних операційних системах (Windows, macOS, Linux) та пристроях (ПК, планшети, смартфони);
- Адаптивний інтерфейс - розміщення елементів має коригуватися залежно від розміру екрану пристрою та вікна браузеру;

- Інтеграція з хмарними сервісами - збереження файлів у хмарних сховищах покращить користувацький досвід;

Функціональні вимоги:

- Використання алгоритмів інтерполяції (зменшення роздільної здатності);
- Налаштування кількості кольорів;
- Можливість вибору адитивних фільтрів (ретро, 8-біт, 16-біт);
- Інструменти для коригування вручну: пензлі, гумка, заливка;
- Експорт у різних форматах (PNG, JPG, GIF, AI);
- Налаштування швидкого доступу до інструментів;

Нефункціональні вимоги:

- Usability - інтуїтивно зрозумілий інтерфейс для швидкого освоєння;
- Швидкість - обробка зображень у режимі реального часу;
- Надійність - стійкість до помилок при завантаженні великих зображень;
- Безпека - шифрування збережених файлів, захист від несанкціонованого доступу;

Запровадження запропонованих технічних вимог дозволить подолати основні проблеми автоматизації піксель-арту. Адаптивні алгоритми зменшення роздільної здатності, інтеграція кольорових палітр і гнучкі параметри експорту мінімізують потребу у ручному редагуванні та підвищать продуктивність, а оптимізація веб-середовища покращить швидкість, роблячи редактор більш доступним і ефективним. Реалізація шифрування файлів і стійких до помилок алгоритмів обробки великих зображень забезпечить надійність і захист даних, а обробка в режимі реального часу підтримає безперервний творчий процес без затримок.

Важливо підкреслити, що запропоновані функціональні та нефункціональні вимоги взаємодоповнюють одна одну: алгоритми інтерполяції й налаштування кількості кольорів забезпечують точну відповідність стилю піксель-арту, тоді як адаптивний інтерфейс і кросплатформеність гарантують однаково позитивний користувацький досвід на будь-якому пристрої.

У результаті отримуємо сучасне гнучке рішення, що спрощує процес перетворення зображень у піксель-арт і відповідає потребам художників, розробників і дизайнерів.

1.4 Постановка задачі

Метою дипломної роботи є розробка сучасної та якісної веб-орієнтованої автоматизованої системи для перетворення зображень у піксель-арт.

Основні завдання:

- Реалізувати механізм автоматизації процесу перетворення зображень у піксель-арт із використанням сучасних алгоритмів кластеризації кольорів і зменшення роздільної здатності;
- Забезпечити можливість ручного редагування результату після автоматичного перетворення для збереження творчого контролю;
- Створити інтуїтивно зрозумілий інтерфейс з підтримкою:
  - кольорових палітр;
  - налаштувань гарячих клавіш;
  - масштабування зображень без втрати якості;
- Інтегрувати підтримку популярних форматів файлів (JPEG, PNG, SVG);
- Забезпечити високу продуктивність додатку у веб-середовищі;

Очікувані результати:

Результатом виконання поставлених задач стане веб-додаток, що дозволяє користувачам ефективно перетворювати зображення у стиль піксель-арту, мінімізуючи потребу в ручній роботі та забезпечуючи високий рівень інтерактивності інтерфейсу. Такий інструмент сприятиме вдосконаленню процесу створення піксель-арту і відповідатиме сучасним вимогам користувачів.

Висновки до розділу 1

У першому розділі дипломної роботи було проаналізовано предметну область, було охоплено історію розвитку піксель-арту, його ключові особливості і застосування у сучасному цифровому просторі. Також було розглянуто основні

переваги та недоліки цього стилю та визначено проблеми, що виникають при створенні піксельної графіки.

Аналіз існуючих популярних програмних рішень, таких як: Aseprite, Piskel, Grafx2 та Adobe Photoshop - показав, що хоча ці інструменти й пропонують значний функціонал для ручного створення піксель-арту, вони не мають повноцінних засобів автоматизації процесу перетворення зображень у піксельний стиль. Саме це підкреслює потребу в розробці спеціалізованої веб-системи яка б значно спростила цей процес та зробила його доступнішим для широкого кола користувачів.

Виявлені основні проблеми:

- Високі трудові витрати при створенні піксель-арту вручну;
- Відсутність інструментів автоматичного зменшення роздільної здатності із збереженням важливих деталей;
- Відсутність інтегрованих механізмів для автоматизованої стилізації графічних елементів;

На основі отриманих даних були сформульовані основні вимоги до веб-системи автоматизованого перетворення зображень у піксель-арт:

- Використовувати алгоритми адаптивного зменшення роздільної здатності, що дозволяють зберегти ключові деталі. Створити функції ручного налаштування рівня деталізації та кількості кольорів для самостійної регуляції якості піксельної графіки;
- Реалізувати інструменти інтерактивного редагування після автоматичного перетворення аби максимально забезпечити користувацький контроль над кінцевим результатом;
- Організувати підтримку сучасних форматів файлів, таких як: PNG, JPEG та SVG;

Проведений аналіз підтверджує актуальність розробки веб-системи для автоматизованого перетворення зображень у піксель-арт. Така система дозволить значно підвищити ефективність роботи дизайнерів, розробників та художників, зменшити витрати часу на створення піксельної графіки і розширити можливості

автоматизованого стилізування зображень. Результати дослідження ляжуть в основу наступних етапів розробки та реалізації відповідної системи.

Розділ 2. МЕТОДОЛОГІЯ ТА ВИБІР АЛГОРИТМІВ

2.1 Огляд методів зменшення роздільної здатності

Зменшення роздільної здатності є ключовим етапом у створенні піксель-арту, оскільки вона безпосередньо деформує і стилізує зображення. У сучасних цифрових зображеннях роздільна здатність - це показник щільності пікселів на одиницю площі - дюйм поверхні(ppi), що застосовується при введенні або виведенні графіки. Чим більше пікселів на дюйм, тим вища роздільна здатність, яка відповідно дозволяє передавати більш дрібні деталі та плавні переходи кольорів.[7]

Однак для піксель-арту така деталізація є надлишковою та навіть шкідливою, оскільки вона суперечить його принципам. Цей стиль насамперед характеризується використанням обмеженої кількості пікселів для передачі образів, що надають йому унікального ретро-естетичного вигляду. Зменшуючи роздільну здатність, художники спрощують деталі, акцентуючи увагу на основних формах та контурах, та створюють унікальні роботи в цьому стилі.

Зменшення роздільної здатності дозволяє:

- спрощувати зображення, роблячи його більш виразним та легким для сприйняття;
- підкреслювати характерну для піксель-арту піксельну структуру;
- відтворювати ретро-естетику обмеженого апаратного забезпечення;

Проте, масштабування зображення без належної обробки може призвести до втрати чіткості та появи небажаних артефактів. Для різних типів графіки масштабування реалізується різними алгоритмами. Векторні зображення при масштабуванні не втрачають якості зображення, а от при збільшенні або зменшенні растрових зображень можуть з’являтися небажані дефекти, а саме: викривлення дрібних деталей або поява помилок в текстурах.

Тому для коректної зміни розмірів растрових зображень використовуються спеціалізовані алгоритми масштабування, що вирівнюють небажані ефекти. [8]

Оскільки піксельний стиль має низьку роздільну здатність, необхідне ретельне розміщення окремих пікселів, ще й доволі часто з обмеженою палітрою кольорів. Орієнтованість спрямована здебільшого на окремі візуальні сигнали. Таким чином визначаються складні фігури з малою роздільною здатністю, аж до окремих пікселів. Процес потребує якісної обробки через що масштабування піксельних зображень є складною проблемою.

Тому для роботи з перетворенням в піксельну графіку рекомендується використовувати методи масштабування, спеціально розроблені для піксель-арту, які зберігають чіткість та структуру пікселів. Існує декілька алгоритмів зменшення роздільної здатності та масштабування, кожен з яких по-різному впливає на кінцевий результат.

2.1.1 Інтерполяція методом найближчого сусіда

Інтерполяція методом найближчого сусіда (рис. 2.1) є одним із найпростіших та найшвидших способів змінювати роздільну здатність зображень. Ідея цього алгоритму проста: спершу будується нова «решітка» пікселів. Для кожної точки цієї решітки визначається, який піксель оригінального зображення розташований до неї найближче. Колір знайденого пікселя без жодних обчислень копіюється у відповідну позицію решітки. У результаті отримується масштабоване зображення, створене простим дублюванням найближчих кольорів, без усереднень або згладжувань. [9]

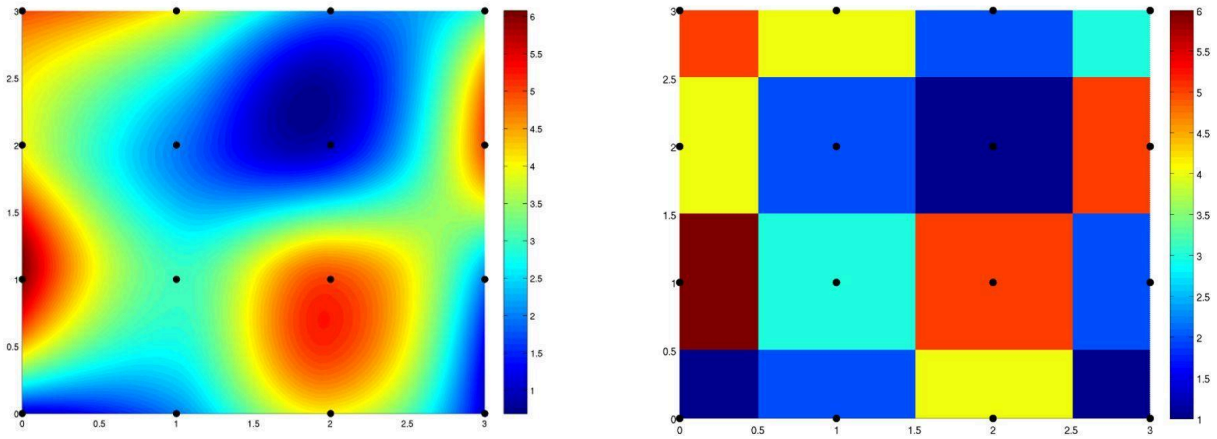


Рис. 2.1. Результат інтерполяції фрагменту зображення методом найближчого сусіда

Застосування інтерполяції методом найближчого сусіда при зменшенні роздільної здатності має свої особливості. Цей метод не додає нових кольорів і не створює згладжування - це дозволяє зберегти чіткість та контрастність зображення. Однак, через простоту алгоритму, можуть виникати такі артефакти як от "сходинки" (рис. 2.2) на діагональних лініях або кривих, це потенційно погіршує загальну яскравість та призводить до втрати великої кількості інформації яка має формувати вихідне чітке зображення (дані не з усіх вихідних пікселів використовуються, якщо зображення зменшене, а при розтягу, деякі дані можуть дублюватися і використовуватися непропорційно більше аніж інші).

Рис. 2.2. Приклад погіршення загальної яскравості зображення та втрати інформації після застосування методу найближчих сусідів

Але, незважаючи на це, метод найближчого сусіда широко використовується завдяки своїй швидкості та низьким обчислювальним витратам. Його особливо зручно використовувати якщо необхідно обробляти зображення в режимі реального часу або для попереднього перегляду редагованих зображень.

У підсумку, інтерполяція методом найближчого сусіда є дуже зручною у цифровій обробці зображень, особливо коли необхідна швидка зміна роздільної здатності з мінімальними обчислювальними витратами. Хоча метод і має свої недоліки - його простота та ефективність забезпечують йому широке застосування в різних галузях обробки зображень.

2.1.2 Білінійна інтерполяція

Ще один простий метод інтерполяції - це білінійна інтерполяція, незначне покращення алгоритму найближчого сусіда яке забезпечує баланс між якістю та обчислювальною ефективністю. Цей метод використовується для збільшення або зменшення розміру зображення шляхом обчислення значень нових пікселів на основі

вагового середнього чотирьох найближчих сусідніх пікселів оригінального зображення (рис.2.3). На відміну від інтерполяції методом найближчого сусіда, білінійна інтерполяція враховує інформацію про інтенсивність кольору кожного з сусідніх пікселів. Так вдається досягти більш плавних переходів та зменшити ефект "сходинок" на діагональних лініях. [10]

Процес білінійної інтерполяції складається з двох послідовних лінійних інтерполяцій: спочатку в одному напрямку (наприклад, по осі x), а потім в іншому (по осі y). Для кожного нового пікселя визначається його положення відносно сусідніх пікселів оригінального зображення, після чого обчислюється вагове середнє значення інтенсивностей цих пікселів.

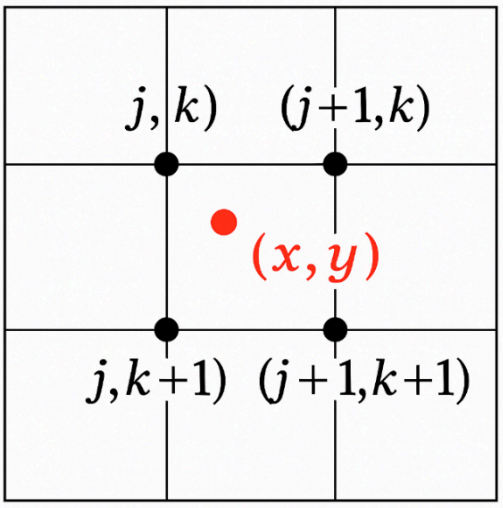


Рис. 2.3. Обчислення значення кольору в точці (x,y) методом білінійної інтерполяції

У білінійній інтерполяції піксель у оригінальному зображенні (позначається як P) на початку відображається на відповідній позиції у вихідному зображенні. Далі оцінюється вплив чотирьох найближчих пікселів навколо заданої точки (позначаються як A, B, C, D). Відстань до P визначає вагу впливу кожного з сусідів на кінцеве значення P. Чим менша відстань, тим більший вплив та вага відповідного пікселя.[11]

На рисунку 2.4 представлено принцип білінійної інтерполяції. Координати пікселів A, B, C, D визначають так: (i, j), (i, j+1), (i+1, j), (i+1, j+1),а координати P - як (u, v).

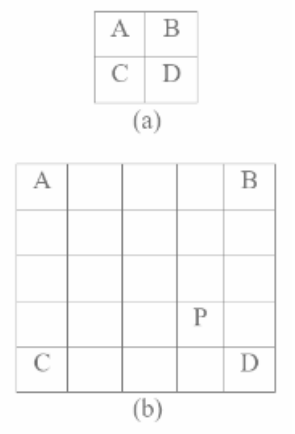


Рис. 2.4. Принцип збільшення зображення завдяки білінійної інтерполяції. (a - початкове зображення, b - збільшене зображення)

Процес білінійної інтерполяції для пікселя P включає три основні кроки:

Крок 1: обчислюється вплив A і B та позначається як E.

$$f(i,j+v) = [f(i, j + 1) - f(i, j)]\ v + f(i, j)$$

Крок 2: обчислюється вплив C і D та позначається як F.

$$f(i+1,j+v) = [f(i+1,j+1) - f(i+1,j)]v + f(i+1,j)$$

Крок 2: обчислюється вплив E і F та позначається як P.

$$f(i+u,j+v) = (1-u)(1-v)f(i,j) - (1-u)vf(i,j+1) + u(1-v)f(i+1,j) + uvf(i+1,j+1)$$

Але при масштабуванні зображень цим алгоритму є один недолік: при збільшенні в n разів зображення розміром W(width) на H(height) пікселів в результаті буде отримано зображення розміром не n*W на n*H пікселів, а (n(W-1)+1) на (n(H-1)+1) пікселів, з доданим ефектом розмиття зображення. Причиною цього є те, що для останнього пікселя вихідного зображення немає пари з якою можна було б провести інтерполювання. [12]

Що стосується зменшення роздільної здатності, білінійна інтерполяція використовується для обчислення значень пікселів у новому, меншому зображенні на основі відповідних областей оригінального зображення. Так зберігається загальна структура та кольорова палітра. Однак, усереднення значень пікселів призводить до артефактів накладання спектру, через що можливе розмиття деталей, що слід

враховувати при обробці зображень, де важлива висока чіткість. Приклад застосування білінійної інтерполяції представлено на рисунку 2.5.

Рис. 2.5. Вхідні дані та результат білінійної інтерполяції

Білінійна інтерполяція є ефективним методом зміни роздільної здатності зображень, який забезпечує баланс між якістю та швидкістю обробки. Не зважаючи на те, що цей метод може створювати незначне розмиття деталей при зменшенні розміру зображення, він широко використовується завдяки своїй простоті та обчислювальній ефективності.

2.1.3 Бікубічна інтерполяція

Бікубічна інтерполяція є одним із найпоширеніших методів зміни роздільної здатності зображень що забезпечує високу якість результату завдяки використанню кубічних поліномів для інтерполяції значень пікселів.

На відміну від білінійної інтерполяції, яка враховує лише 4 сусідні пікселі (матриця 2x2), бікубічна інтерполяція використовує більш широку решітку з 16 пікселів, розташованих навколо точки - пікселя, для якого необхідно визначити нове значення. Процес включає обчислення вагових коефіцієнтів для кожного з оригінальних пікселів залежно від їх відстані до нового пікселя, після чого обчислюється зважене середнє значення. [13]

Основна мета цього методу - створити кубічну поліноміальну функцію, що забезпечує плавний перехід між пікселями шляхом зниження ризику виникнення різких перепадів у значеннях. (рис. 2.6)

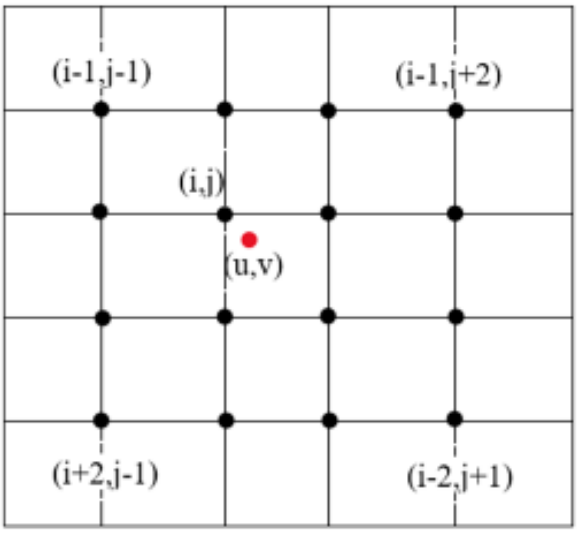


Рис. 2.6. Обчислення значення кольорув точці (x,y) методом бікубічної інтерполяції

Для досягнення цієї плавності бікубічна інтерполяція включає кілька етапів:

Крок 1: Визначається 4x4 рещітка з найближчих пікселів навколо цільової точки (u, v), яка є центром цієї сітки.

Крок 2: Як і в білінійній інтерполяції розраховується середнє вагове цих пікселів, про те в цьому випадку 16-ти. Вага кожного пікселя залежить від його відстані до точки (u, v). Чим ближче до цільової точки - тим більший вплив на остаточне значення.

Крок 3: Визначення значення нового пікселя в цільовій точці (u, v) відбувається

шляхом обчислення кубічного полінома на основі відповідних координат.

При зменшенні роздільної здатності зображення за допомогою бікубічної інтерполяції відбувається усереднення значень пікселів на основі їх оточення. Таким способом зберігається більшість деталей та уникається поява артефактів. Цей метод особливо ефективний зображень з плавними переходами кольорів, оскільки забезпечує високу якість зменшеного зображення без втрати важливих деталей.

Проте, подібно до ситуації з білінійною інтерполяцією, існує велика вірогідність створення ефекту розмиття, що в свою чергу не є бажаним для стилю

піксель-арт, для якого важлива саме чіткість відображення і висока контрастність. Результат бікубічної інтерполяції розглянемо на рис. 2.7.

Рис. 2.7. Вхідні дані та результат бікубічної інтерполяції

2.1.4 Аналіз методів масштабування

Зменшення роздільної здатності є основним етапом у перетворенні зображень у піксель-арт, оскільки дозволяє спростити форму об'єктів та адаптувати їх до стилізованої сіткової структури. Аналіз методів показав що вибір алгоритму масштабування є критично важливим для досягнення відповідного художнього ефекту без великої втрати деталей або небажаних спотворень контурів.

Метод найближчого сусіда є найпростішим варіантом зменшення роздільної здатності, оскільки не додає нових кольорів і забезпечує чіткість контурів. Проте має суттєві недоліки, зокрема втрату деталей та ефект "сходинок" на діагональних лініях.

Білінійна інтерполяція виконує збільшення завдяки лінійному взаємозв'язку між чотирма найближчими пікселями навколо цільової точки та дозволяє створювати згладжені переходи кольорів, зменшуючи артефакти. Цей метод швидко обчислюється, через це є ідеальним для застосувань, де швидкість є критичною. Однак, білінійна інтерполяція все ж може призводити до втрати деталей та введення артефактів таких як розмиття деталей, що в свою чергу не є бажаним для піксель-арту.

Бікубічна інтерполяція є більш точним методом, оскільки використовує кубічні поліноми для визначення значень нових пікселів. Перехід кольорів виглядає гладко та більш природньо. Наочне представлення різниці зміни кольорів при використанні білінійної та кубічної інтерполяцій зображено на рисунку 2.8.

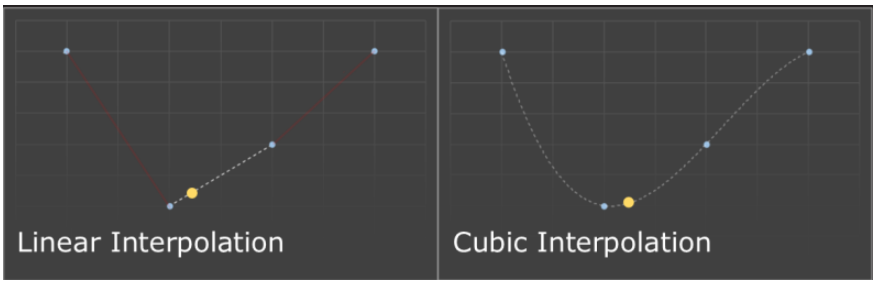


Рис. 2.8. Порівняння тенденцій зміни кольорів за допомогою білінійної(зліва) та бікубічної(справа) інтерполяцій

Бікубічна інтерполяція розраховує шістнадцять найближчих пікселів навколо цільової точки, тому забезпечує вищу якість зображення з кращим збереженням деталей та меншою кількістю артефактів. Однак, це означає, що бікубічна інтерполяція є найвимогливішою серед розглянутих методів до обчислювальних ресурсів і може бути значно повільнішою в порівнянні з тією ж білінійною інтерполяцією. (рис. 2.9).

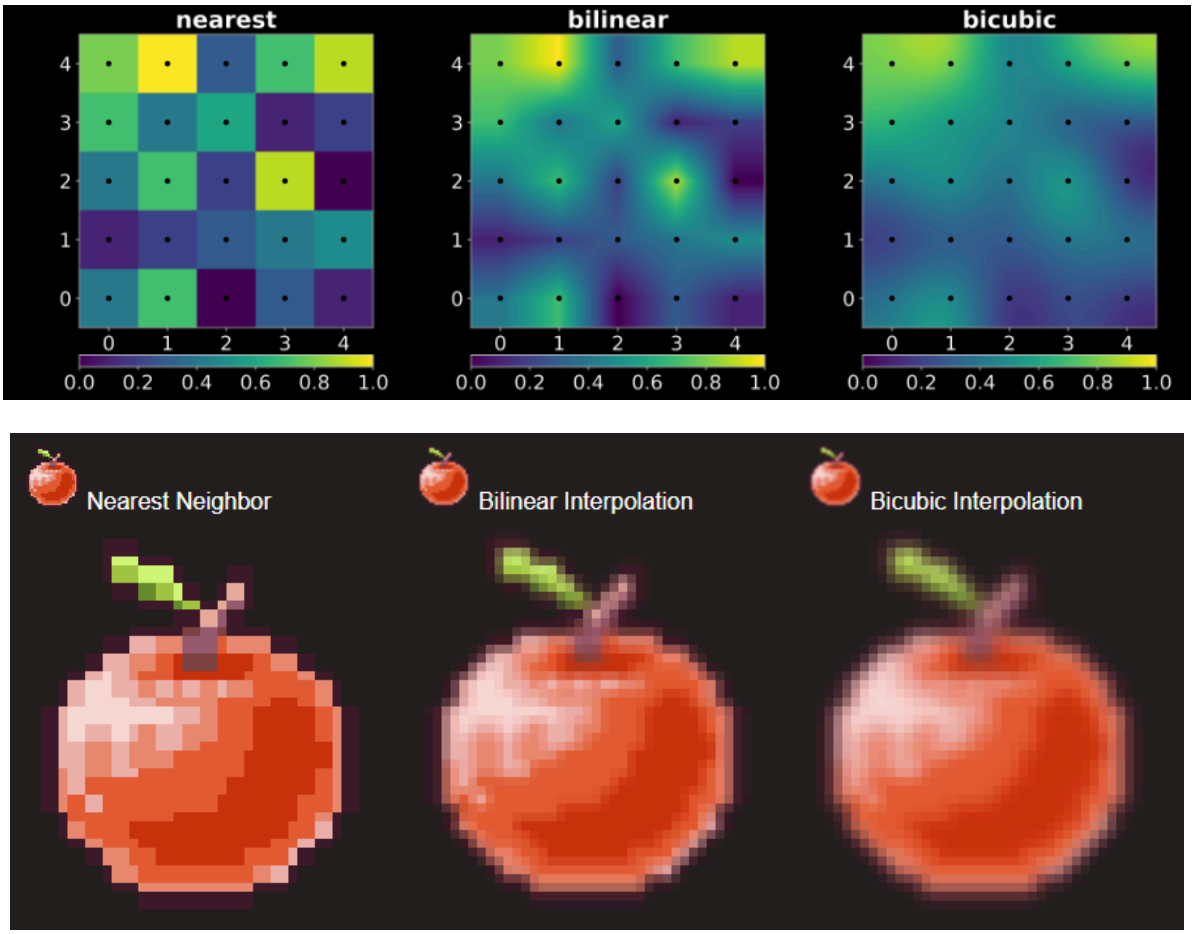


Рис. 2.9. Представлення результатів обробки зображень методами найближчого сусіда, білінійної та бікубічної інтерполяцій

2.1.5 Вибір оптимального методу зменшення роздільної здатності для веб-редактору

Розглянуті методи надають уявлення про основні тенденції розвитку алгоритмів масштабування зображень. Для реалізації веб-додатку оптимальним методом є «інтерполяція методом найближчого сусіда», адже вона достатньою мірою забезпечує збереження чітких контурів, що є критично важливим для піксель-арту.

Основні переваги методу найближчого сусіда:

- Зберігає чітку піксельну структуру без розмиття та згладжування, як у білінійній чи бікубічній інтерполяції;
- Необхідні мінімальні обчислювальні витрати та ресурси, що робить його ідеальним для веб-додатків у реальному часі;
- Зберігає контрастність завдяки відсутності змішування кольорів і забезпечує автентичний вигляд піксель-арту;

Недоліком є можливі "сходинки" на діагональних лініях, проте для піксель-арту це часто не є критичним, оскільки відповідає ретро-естетиці. У разі потреби додаткової обробки можна застосувати фільтри для підсилення контурів або ж використовувати ручні інструменти обробки зображення.

Таким чином, метод найближчого сусіда є найкращим вибором для веб-додатку перетворення зображень у піксель-арт, оскільки поєднує високу швидкість обробки та збереження характерного стилю піксельної графіки.

2.2 Кластеризація кольорів

Кластеризація кольорів є наступним ключовим етапом обробки зображень у піксель-арт. Вона покращує сприйняття цього стилю, усуваючи зайвий «шум» та створюючи єдину гармонійну палітру. Зменшення кількості кольорів у зображенні робить його візуально цілісним і спрощує подальше редагування, що особливо важливо в контексті стилізованої графіки. Використання кластеризації мінімізує переходи між відтінками, зменшуючи кількість проміжних кольорів і підкреслюючи основні елементи композиції.

Впровадження кластеризації кольорів у процес генерації піксель-арту дозволяє автоматизованим системам генерувати більш природні та естетично привабливі результати, що робить цей метод невід'ємною частиною створення високоякісного стилізованого контенту у стилі пікселю.

2.2.1 Алгоритм K-Means

Алгоритм K-Means це один із найпростіших алгоритмів з простою процедурою класифікації заданих наборів. Він групує дані та розділяє вибірки на n груп рівної дисперсії (рис. 2.11), мінімізуючи критерій, відомий як інерція або сума квадратів у межах кластера. Головна мета цього методу - розділити набір даних на k окремих кластерів на основі подібності точок даних. У результаті отримується розділений набір з n вибірок x на k непересічних кластерів, кожен з яких описується середнім значеннями які зазвичай називають «центроїдами» кластера; (рис. 2.10)

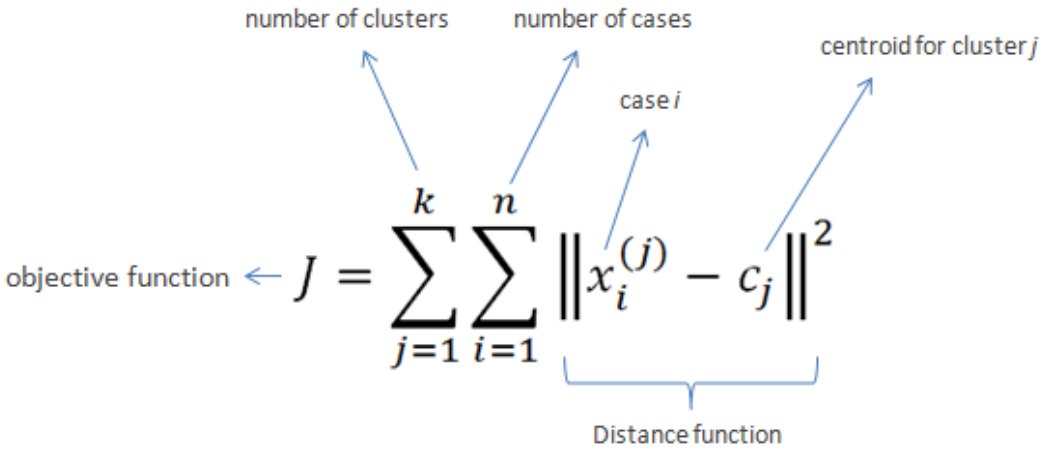


Рис. 2.10. Представлення формули алгоритму K-Means

Алгоритм складається з трьох кроків:

Крок 1: Обираються початкові центроїди. Після ініціалізації вони складаються з циклу між двома наступними кроками. Спочатку призначається кожен зразок до найближчого центроїда.

Крок 2: Потім створюються нові центроїди на основі середніх значень зразків, що були призначені кожному попередньому центроїду в попередньому кроці..

Крок 3: Обчислюється різниця між старим і новим центроїдами. Зрештою алгоритм повторює цикл допоки значення кількості кольорів не стане меншим за вказане порогове значення. [14]

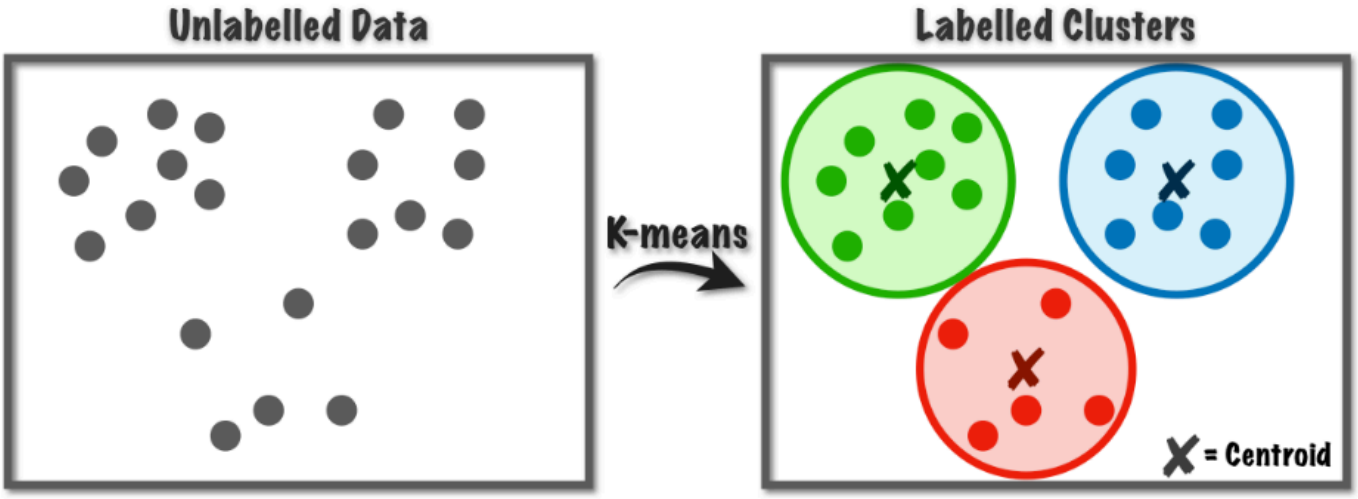


Рис. 2.11. Принцип роботи алгоритму K-Means

Алгоритм K-середніх широко використовується в комп'ютерній графіці для обробки та аналізу зображень. Одним із ключових застосувань є кластеризація кольорів яка дозволяє зменшити кількість кольорів у зображенні, зберігаючи його візуальну якість.

Працює він шляхом представлення кожного пікселя як точки в колірному просторі, наприклад, у форматі RGB, де кожен колір має три координати, що відповідають інтенсивності червоного, зеленого та синього каналів. Перед початком роботи алгоритму необхідно визначити кількість кластерів k, яка відповідає бажаній кількості кольорів у фінальному зображенні (може бути зазначеною користувачем редактору). Далі здійснюється ініціалізація центроїдів, після чого для кожного кластера обчислюється нове середнє значення кольорів пікселів, що до нього належать. Отримані середні значення використовуються для оновлення положень центроїдів, цей процес повторюється ітеративно, кожного разу перегруповуючи пікселі на основі оновлених центрів кластерів, аж до досягнення збіжності, коли зміни в положеннях центроїдів стають незначними.

Отже, у контексті обробки зображень виконується сегментація за кольорами, тобто розбиття кольорового простору зображення на чітко визначені області з обмеженою палітрою. Зображення спрощується та набуває обрисів стилізації що подібна піксель-арту або іншим художнім обробкам в обмежених палітрах. (рис. 2.12)

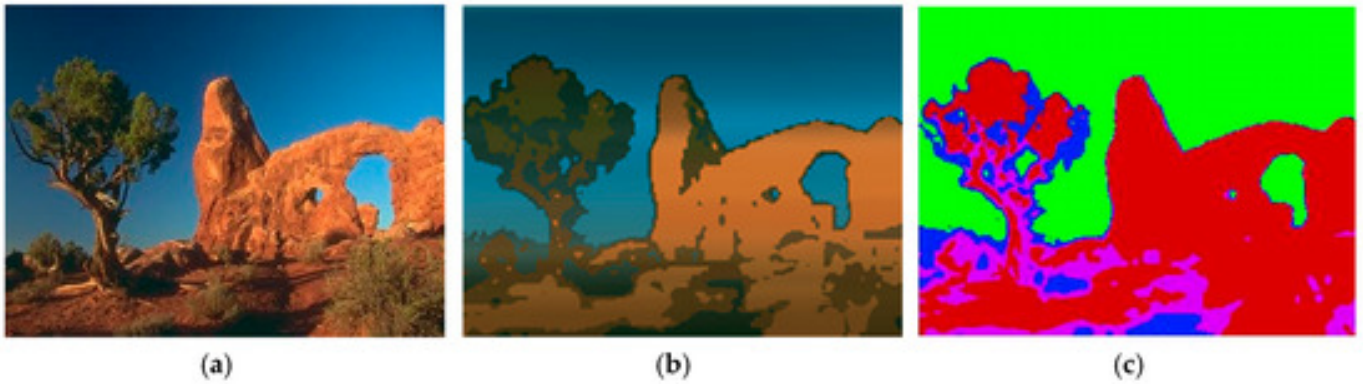


Рис. 2.12. Оригінальне зображення (a); Результат застосування алгоритму K-Means(b); Демонстрація кольорових сегментів зображення(c*)*;

Переваги використання методу K-середніх у кластеризації кольорів [15]*:*

- Алгоритм є ефективним в обчислюваннях і підходить для обробки великих зображень;
- Легко реалізується та інтегрується в існуючі графічні додатки;
- Дозволяє контролювати кількість кольорів у фінальному зображенні шляхом налаштування параметра k;

Обмеження:

- Результати можуть залежати від початкового вибору центроїдів;
- Алгоритм припускає сферичну форму кластерів, що може не відповідати реальним даним;
- Не гарантує досягнення глобального оптимуму;
-

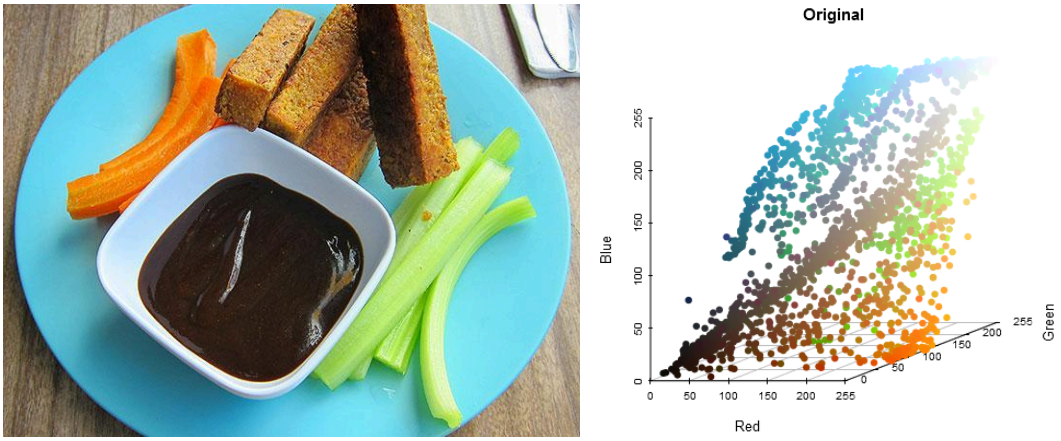
2.2.2 Алгоритм Median Cut

Median Cut - це алгоритм для сортування даних довільної кількості вимірів у ряди наборів шляхом рекурсивного розрізання кожного набору даних у середній

точці вздовж найдовшого виміру. Серединний зріз зазвичай використовується для квантування кольорів. Наприклад, щоб зменшити зображення з 64 тис. кольорів до 256 кольорів, використовується медіанний розріз, щоб знайти 256 кольорів, які добре відповідають вихідним даним. [16]

Реалізація квантування кольорів:

Першим кроком є вибір розміру для розділення. Оскільки необхідно щоб кольори, які найближче один до одного, опинилися в групі, зазвичай обирається розмір із найбільшим діапазоном між мінімальним і максимальним значенням серед пікселів, щоб розділити його на дві частини. Початкова група пікселів розбивається на значення червоного, зеленого та синього каналів, далі обчислюється діапазон значень і середнє значення кожного каналу. (рис. 2.13)



(a) (b)

Рис. 2.13. Оригінальне зображення (a) вибірка із компонентів RGB(b)

Після цього пікселі зі значеннями каналу більшими або меншими за медіану, сортуються у дві нові групи, які замінюють вихідну групу. (рис. 2.14)

Після сортування сегмента він переміщує верхню половину пікселів у новий сегмент (саме цей крок дає назву алгоритму медіанного вирізання: сегменти поділяються надвоє у списку пікселів). Цей процес повторюється до досягнення потрібної кількості в палітрі кольорів. Потім пікселям у кожній групі призначається середній колір групи, і зображення перемальовується. (рис. 2.15)

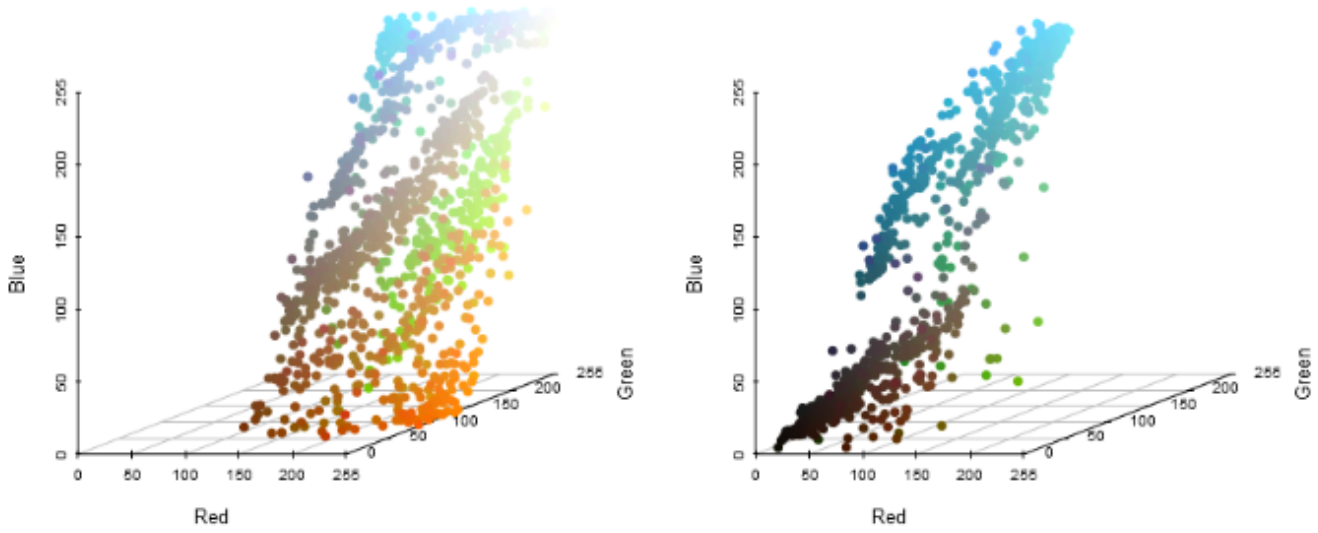


Рис. 2.14. Розділення елементів медіаною

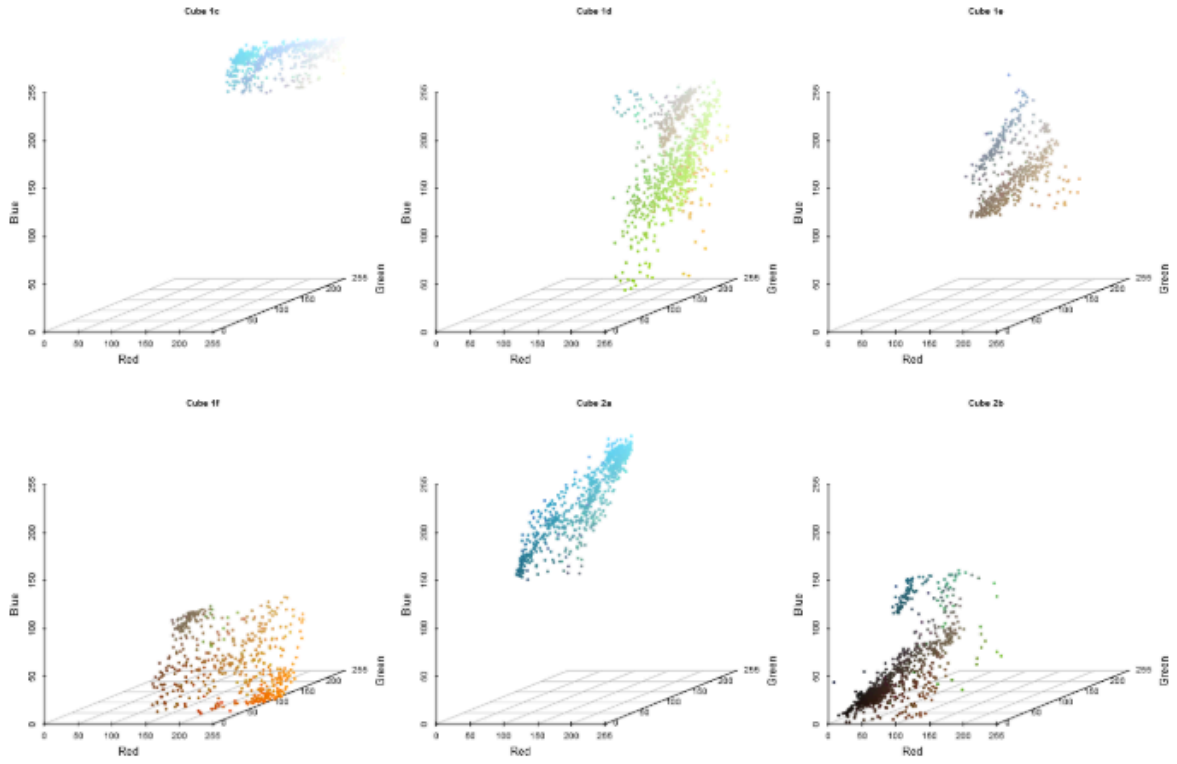


Рис. 2.15. Ітераційне виведення домінуючих кольорів

Результати виконання алгоритму Median Cut представлено на рисунку 2.16.

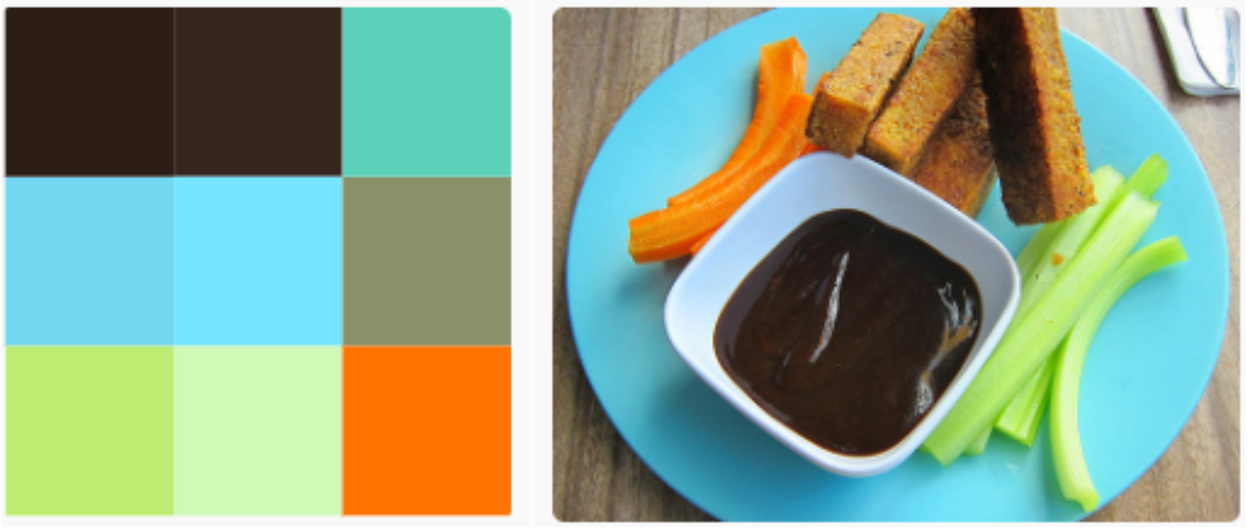


Рис. 2.16. Результат застосування алгоритму Median Cut

Переваги використання Median cut у кластеризації кольорів:

- Алгоритм легко реалізується та не потребує складних обчислень, він також є швидшим за методи, що використовують складні оптимізаційні процедури, такі як K-Means;
- Розбиття відбувається на основі рівномірного розподілу кольорів, що забезпечує збалансовану палітру;
- Він добре працює для зображень із рівномірним розподілом кольорів, забезпечуючи візуально приємні результати;
- Дозволяє точно контролювати кількість кольорів у фінальному зображенні;

Обмеження:

- Рівномірно розрізає простір кольорів, ігноруючи те, що деякі кольори можуть бути більш важливими для зображення;
- Якщо в зображенні багато різноманітних кольорів, які не мають однакового розподілу він може втрачати важливі деталі;
- Може створювати небажані артефакти в областях з великою кількістю рідкісних кольорів;
- У порівнянні з іншими методами, такими як K-Means, може спрощувати переходи між кольорами, роблячи градієнти менш плавними;

2.2.3 Алгоритм Octree Color Quantization

Octree Color Quantization - це метод зменшення кількості кольорів у зображенні, що базується на використанні структури даних, відомої як октодерево (рис. 2.17).

Ця структура є деревом, в якому кожен внутрішній вузол має до восьми нащадків, що дозволяє ефективно представляти тривимірний колірний простір, такий як RGB, де кожна з колірних компонент (червоний, зелений, синій) може приймати 256 значень, що в сумі дає понад 16 мільйонів можливих кольорів. [17]

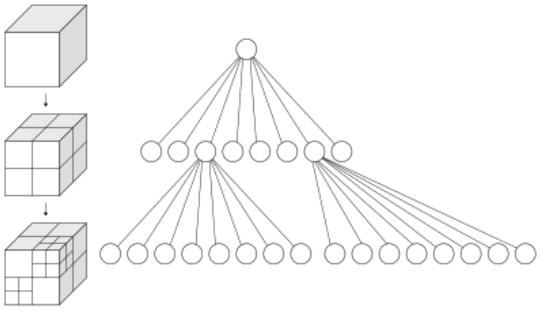


Рис. 2.17. Приклад рекурсивного розбиття куба на октанти і відповідне октодерево

Основні етапи алгоритму:

Крок 1: Будується октодерево. Кожен піксель зображення додається до октодерева, де його колір визначає шлях від кореня до листового вузла. На кожному рівні дерева використовуються відповідні біти колірних компонент для вибору одного з восьми нащадків.

Крок 2: Відбувається обрізка дерева за умов якщо кількість листових вузлів перевищує бажану кількість кольорів (K). Дерево обрізається шляхом об'єднання найменш значущих вузлів шляхом злиття на нижчих рівнях, що мають найменший внесок у загальну колірну палітру.

Крок 3: Після обрізки дерева кольори, представлені листовими вузлами, використовуються для створення кінцевої палітри з K кольорів.

Крок 4: Кожен піксель зображення замінюється найближчим кольором з отриманої палітри, що зменшує загальну кількість кольорів у зображенні. (рис. 2.18).

Переваги використання октодерева у кластеризації кольорів:

- Алгоритм швидко обробляє зображення;
- Можливість точно визначити кількість кольорів у кінцевій палітрі;

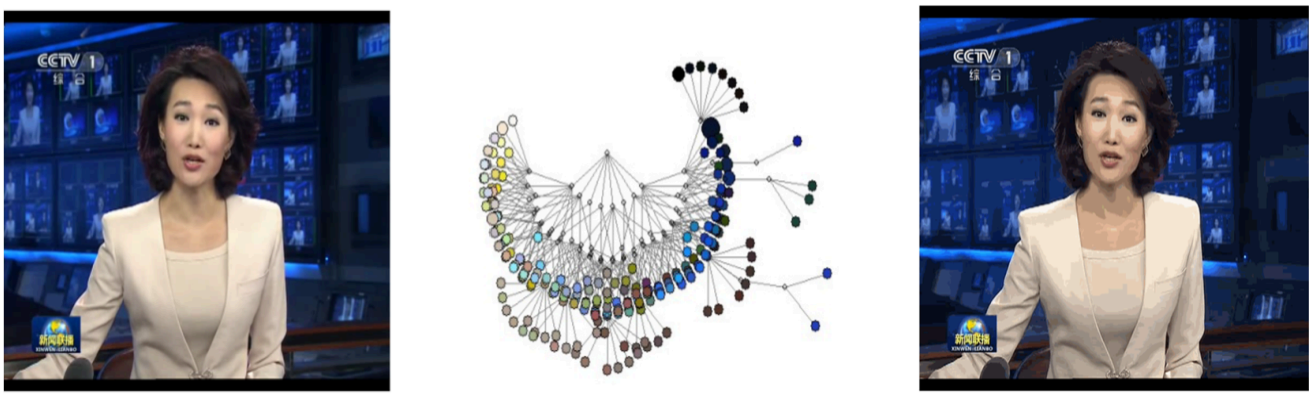


Рис. 2.18. a -оригінальне зображення; b - компоненти октодерева; c - результат застосування алгоритму Octree Color Quantization

Обмеження:

- Алгоритм не враховує частоту появи кольорів у зображенні, через що може виникнути втрата важливих колірних відтінків, особливо якщо зображення містить багато рідкісних кольорів або шумів;
- При значному зменшенні кількості кольорів (малий K) якість зображення може суттєво погіршитися через втрату важливих колірних деталей;

2.2.4 Вибір оптимального алгоритму кластеризації для веб-редактору

Вибір оптимального алгоритму кластеризації для автоматизованого перетворення зображень у піксель-арт залежить від кількох факторів: швидкості обробки, якості отриманої палітри та можливості ефективної роботи у веб-застосунках. Серед розглянутих алгоритмів K-Means, Median Cut і Octree Color Quantization найбільш збалансованими з точки зору швидкодії та якості є Median Cut і Octree Color Quantization, тоді як K-Means забезпечує вищу якість але потребує більших обчислювальних ресурсів. Для веб-застосунків, що працюють у браузері, критично важливою є можливість виконувати кластеризацію кольорів у реальному часі без значного навантаження на процесор користувача.

Для веб-застосунків ключове значення має швидка реакція інтерфейсу в реальному часі. Оптимальною є гібридна схема, у якій основну кластеризацію виконує K-Means, а Median Cut попередньо відбирає базові кольори палітри.

Етапи застосування:

1. K-Means (основний етап) - алгоритм одразу кластеризує всі пікселі оригінального зображення у просторі RGB. Після випадкової ініціалізації центрів він ітеративно:

- призначає кожен піксель найближчому центрові за евклідовою відстанню;
- обчислює центри як середнє значення координат пікселів кластера;
- зупиняється, коли зміщення центрів менше порога збіжності або досягнуто максимальну кількість ітерацій ($t \approx 5\text{-}10$).

2. Median Cut (друга стадія) - працює вже з центрами, отриманими K-Means, і розраховує з них компактну палітру з десяти презентативних відтінків. Цей крок відсікає надлишкові кольори, залишаючи тільки ті, що найточніше описують структуру кластерів.
3. Octree Color Quantization - опціональна альтернатива, якою можна скористатися замість другої стадії, якщо потрібно ще більше скоротити час обробки, але базовим алгоритмом усе одно залишається K-Means.

Отже, запропонована послідовність поєднує точність і швидкодію: K-Means одразу забезпечує детальний розподіл усіх кольорів зображення, а Median Cut після цього швидко стискає результат до десяти ключових відтінків, оптимізуючи пам’ятні витрати та розмір палітри. Така гібридна схема підходить для веб-застосунків, де потрібна одночасно швидка реакція інтерфейсу й високоякісне збереження піксель-арту.

2.3 Види фільтрів для спрощення зображень

Використання фільтрів допомагає спростити зображення, зменшуючи кількість дрібних деталей і шумів, що можуть ускладнювати кластеризацію кольорів. Без

відповідної обробки висока деталізація оригінального зображення може призвести до втрати стилістичної виразності або до того, що алгоритм кластеризації кольорів не зможе коректно адаптувати палітру під потрібний художній стиль.

Фільтри відіграють ключову роль у підготовці зображення до перетворення, дозволяючи виділити основні форми, згладити різкі переходи та підкреслити контури, що є важливим для отримання чистого й виразного піксельного стилю. Крім того, вони можуть зменшити артефакти, викликані цифровою компресією або шумом, що є актуальним для низькоякісних зображень. Отже, правильний вибір фільтрів

перед кластеризацією кольорів і зменшенням роздільної здатності сприяє отриманню якісного зображення, яке виглядає гармонійно в стилістиці піксель-арту.

Гауссове розмиття (Gaussian Blur)

Гауссове розмиття є методом згладжування зображення шляхом застосування гауссового фільтра, який розподіляє ваги пікселів відповідно до гауссової кривої. Це призводить до рівномірного розмиття, зменшуючи шум та дрібні деталі, що може бути корисним для підготовки зображення до подальшої обробки, наприклад, виділення контурів. (рис. 2.19)

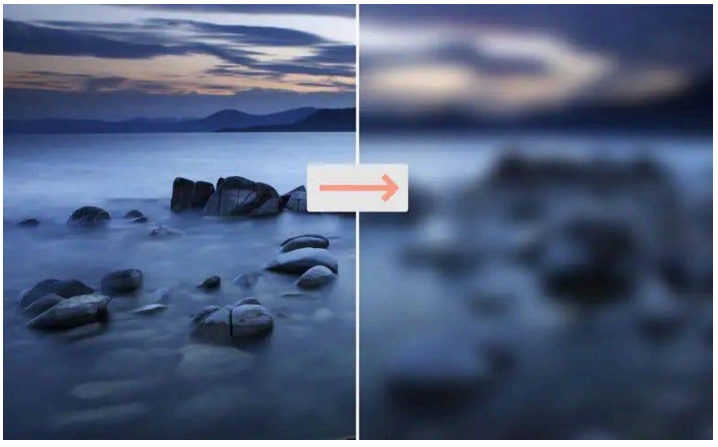


Рис. 2.19. Приклад Гауссового розмиття

Контурне виділення (Sobel Edge Detection)

Метод виділення контурів за допомогою оператора Собеля використовується для виявлення границь об'єктів у зображенні. Оператор обчислює градієнт яскравості в кожній точці зображення, що дозволяє визначити області з різкими змінами інтенсивності, тобто контури. Це корисно для сегментації та аналізу структури зображення. (рис. 2.20)

Рис. 2.20. Приклад контурного виділення

Адаптивне порогове перетворення (Adaptive Thresholding)

Адаптивне порогове перетворення є методом бінаризації зображення, при якому поріг визначається для кожної області окремо на основі характеристик цієї області (рис. 2.21). Це дозволяє ефективно обробляти зображення з нерівномірним освітленням або тінями, забезпечуючи більш точне розділення об'єктів і фону. [18]

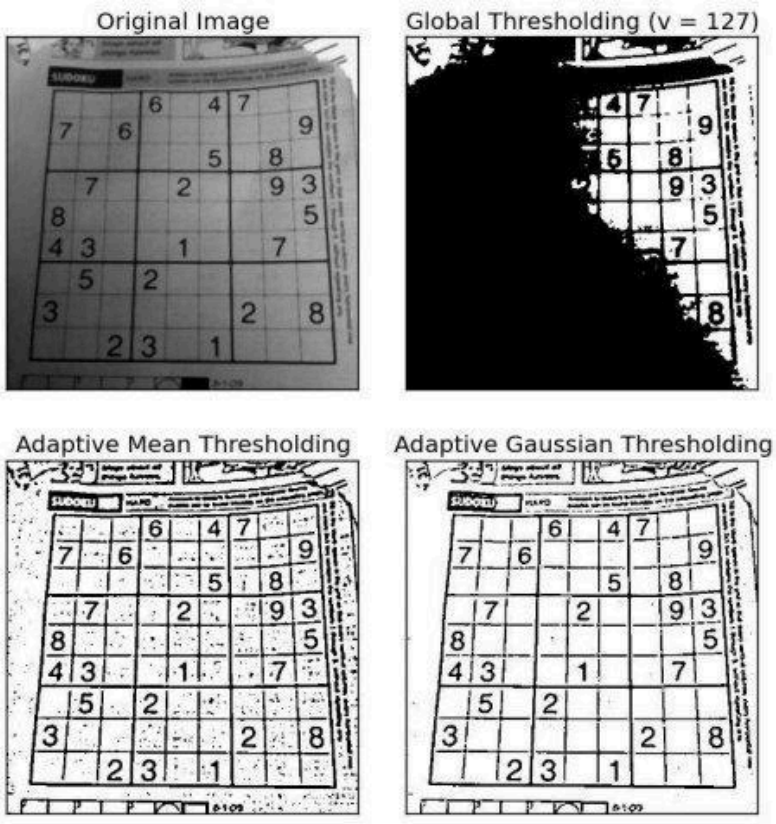


Рис. 2.21. Приклади адаптивного порогового перетворення

Ефект мультфільму (Cartoon Effect)

Ефект мультфільму поєднує кілька методів обробки зображення, включаючи білінійне згладжування та виділення контурів, щоб перетворити фотографію на зображення, схоже на кадр з мультфільму. Це досягається шляхом згладжування текстур та збереження або підсилення контурів, що надає зображенню стилізований вигляд. (рис. 2.22).

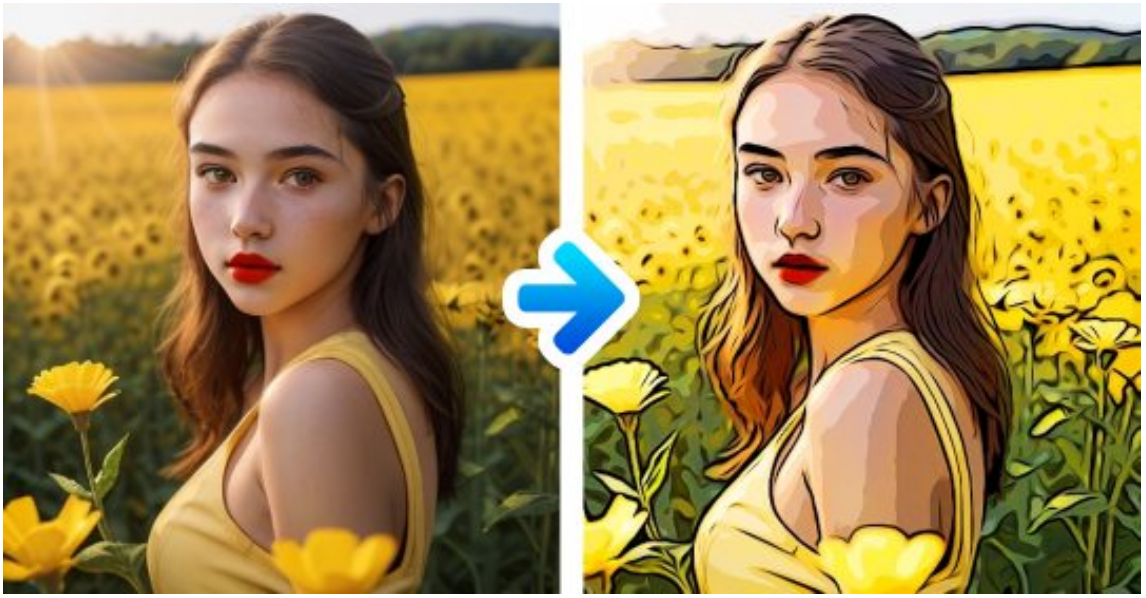


Рис. 2.22. Приклад накладання ефекту мультфільму

Висновки до розділу 2

У цьому розділі було проведено аналіз та обґрунтування вибору алгоритмів, необхідних для автоматизованої веб-редактору зображень у стиль піксель-арту. Основна увага приділялася вибору оптимального алгоритму зменшення роздільної здатності, методу кластеризації кольорів, а також фільтрів попередньої обробки, які дозволяють отримати максимально якісний результат.

Алгоритм зменшення роздільної здатності:

Аналіз існуючих методів масштабування зображень показав, що серед основних підходів (найближчого сусіда, білінійної і бікубічної інтерполяції) найбільш ефективним для завдання генерації піксель-арту є метод найближчого сусіда.

Цей метод не змінює кольорову палітру зображення та дозволяє зберігати чіткі контури об’єктів, що є важливим для стилізації зображення у вигляді піксельної графіки. Білінійна та бікубічна інтерполяції хоча й забезпечують більш плавні переходи між пікселями, проте призводять до згладжування контурів, що є небажаним у контексті піксель-арту.

У результаті аналізу визначено, що у веб-системі для перетворення зображень у піксель-арт буде застосовано саме метод найближчого сусіда.

Метод кластеризації кольорів:

Для стилізації піксель-арту критично важливою є правильна обробка кольорів, оскільки піксельні зображення традиційно мають обмежену кольорову палітру. Аналіз існуючих методів зменшення кількості кольорів показав, що серед найбільш ефективних алгоритмів (K-Means, Median Cut, Octree Color Quantization) найкращим вибором є алгоритм K-Means. Завдяки ньому можна задавати необхідну кількість кольорів/кластерів, що дозволяє адаптувати палітру зображення. Хоча й алгоритм не є найлегшим у реалізації, він забезпечує високу точність кольорової кластеризації, а також у порівнянні з Median Cut та Octree, K-Means краще адаптується до реальних колірних особливостей зображення. Зважаючи на це, для кластеризації кольорів у веб-додатку буде використано K-Means, який дозволить досягти якісного стилізованого ефекту з можливістю налаштування кількості кольорів для отримання потрібної палітри.

Фільтри попередньої обробки зображень:

Перед кластеризацією кольорів та зменшенням роздільної здатності важливо підготувати зображення, щоб воно краще піддавалося стилізації у піксель-арт. Для цього використовуються фільтри, які дозволяють оптимізувати зображення, підкреслити його ключові деталі або, навпаки, усунути дрібні шуми.

У результаті дослідження було визначено, що доцільним є застосування як згладжуючих, так і контуропідсилюючих фільтрів, що разом дозволяють отримати більш чисте та виразне зображення для подальшої обробки. Зокрема, фільтри згладжування допомагають усунути надлишкові деталі та зменшити шум, що дозволяє отримати більш рівномірні кольорові області після кластеризації. Фільтри підсилення контурів, у свою чергу, забезпечують чіткість границь між об’єктами, що є важливим для збереження стилістики піксельного мистецтва.

Отже, усі розглянуті фільтри є ефективними інструментами для покращення вхідних зображень та можуть використовуватися в залежності від особливостей початкового матеріалу та бажаного ефекту. Їх комплексне застосування дозволяє досягти оптимального балансу між деталізацією та спрощенням структури зображення перед його конвертацією у піксель-арт.

В результаті аналізу та тестування методів, що можуть бути використані для автоматизованого перетворення зображень у піксель-арт, було визначено такі ключові рішення для реалізації веб-додатку:

1. Метод масштабування: інтерполяція методом найближчого сусіда.
2. Метод кластеризації кольорів: K-Means.
3. Попередня обробка: поєднання всіх фільтрів.

Використання цих методів у веб-редакторі забезпечить високу якість автоматизованого перетворення зображень у стиль піксель-арту, дозволяючи користувачам отримувати результат, максимально наближений до ручного малювання, але значно швидше та без потреби у спеціальних навичках редагування графіки.

Розділ 3. ПРОЄКТУВАННЯ ТА РЕАЛІЗАЦІЯ АВТОМАТИЗОВАНОЇ ВЕБ-СИСТЕМИ

3.1 Огляд засобів і методологій розробки

Для успішної реалізації проєкту необхідний обґрунтований вибір технологічних засобів та чітка методологія розробки, що дозволяють забезпечити не лише коректне виконання поставлених завдань, але й подальшу підтримку та розвиток системи.

У цьому розділі визначається склад технологічного стеку: мов програмування, що використовуються для побудови веб-інструменту перетворення зображень у піксель-арт, та відповідної моделі життєвого циклу проєкту, яка надасть чітку структуру дій задля швидкої та якісної реалізації.

Це поєднає високі вимоги до продуктивності й масштабованості з простотою супроводу коду в реальному часі та гарантує своєчасне виявлення й усунення потенційних ризиків під час розробки.

3.1.1 Огляд мов програмування, фреймворків та бібліотек

- HTML5

HTML5 - це п’ята основна версія стандарту мови розмітки веб-сторінок, яка у випадку веб-редактора зображень вводитиме низку семантичних тегів (<header>, <section>, <footer>) та мультимедійних елементів (<video>, <audio>), а також ключовий <canvas> для растрової графіки. Цей елемент слугуватиме каркасом «полотна», на якому мова, що реалізує функціонал застосунку, безпосередньо читатиме і записуватиме пікселі для масштабування, кластеризації кольорів та інших обчислень на рівні кожного пікселя. [19]

- CSS3

CSS3 - модульна третя версія каскадних таблиць стилів, що розширює можливості оформлення за допомогою таких технологій:

- Flexbox, для одновимірного розподілу простору та вирівнювання елементів уздовж однієї осі;
- Grid, для створення двовимірних сіток із тонким керуванням рядками та стовпцями;
- Анімації, переходи й трансформації для динамічного інтерфейсу.

Ці модулі створюватимуть адаптивну та зручну панель інструментів, модальні вікна й інші UI-компоненти, що коректно відображатимуться на різних екранах і в різних браузерах. [20]

- Vanilla JavaScript

Vanilla JavaScript означає використання мови JavaScript без жодних фреймворків чи бібліотек. Завдяки цьому мінімізується обсяг коду та стає легше уникнути сторонніх абстракцій і досягти максимальної швидкодії клієнтської частини.

З огляду розробки веб-редактора JS реалізовує: логіку завантаження зображень, керування інструментами (пензлик, гумка, фільтри, тощо), історію подій, імпорт та експорт зображень. [21]

- Canvas API

Canvas API - це інтерфейс програмування, який надає можливість за допомогою JavaScript малювати та маніпулювати растровою графікою в елементі <canvas>. З його допомогою можна:

- отримувати та змінювати піксельні дані;
- застосовувати фільтри (Наприклад Gaussian blur, Sobel edge detection, posterize та ін.);
- виконувати анімацію і реагувати на події користувача.

У проєкті автоматизованої системи перетворення зображення Canvas API є полем рендерингу пікселізації, адже він забезпечує високу точність маніпуляцій пікселями та дозволяє реалізувати всі ключові алгоритми обробки зображень прямо в браузері в режимі реального часу. [22]

Обраний технологічний стек: поєднання HTML5, CSS3, чистого JavaScript і Canvas API - забезпечить оптимальний баланс між продуктивністю, гнучкістю та простотою підтримки. Така комбінація інструментів гарантує швидку роботу додатку, просту інтеграцію та подальший розвиток функціоналу.

3.1.2 Вибір життєвого циклу проєкту

Життєвий цикл проєкту (Software Development Life Cycle, SDLC) - це сукупність фаз, через які проходить розробка продукту: від збору вимог до експлуатації та підтримки. Вибір моделі SDLC визначає як організовувати планування, оцінку ризиків, розробку, тестування і презентацію результатів.

У цьому проєкті обрана спіральна модель (Spiral Model) - це модель життєвого циклу розробки програмного забезпечення яка має в собі систематичний та ітеративний підхід. ЇЇ схематичним зображенням є спіраль з великою кількість витків. Точна кількість петель спіралі відразу не відома і може змінюватися в залежності від проєкту. Кожен виток спіралі називається фазою процесу розробки програмного забезпечення. [23]

Деякі особливості в етапах спіральної моделі:

- Точну кількість фаз/етапів, необхідних для розробки продукту, може будь коли змінювати керівник проекту залежно від ризиків проєкту та загальної ситуації під час розробки;
- Вона базується на ідеї спіралі, кожна ітерація якої представляє повний цикл розробки програмного забезпечення: від збору та аналізу вимог до проектування, впровадження, тестування і підтримки. В результаті кожного циклу отримується прототип проєкту.

Переваги та недоліки Spiral Model представлені у таблиці 3.1.

Для веб-редактору ця модель є доцільним вибором адже:

1. Кожен цикл дозволяє виявити і пом’якшити ризики (технічні, часові, ресурсні) перед переходом на наступний етап;
2. Є можливість коригувати функціонал і архітектуру після кожної ітерації;

Кожний виток завершується робочим прототипом, відбувається демонстрація прогресу та збір зворотного зв’язку від користувачів.

Таблиця 3.1

Переваги та недоліки Spiral Model

Переваги	Недоліки
⇒ Якісний та грунтовний аналіз ризиків;	⇒ Дорога у використанні;
⇒ Поетапне створення робочих прототипів програмного продукту.	⇒ Вимагає залучення експертів задля якісної оцінки ризиків;
⇒ Можливість внесення змін і додавання нової функціональності на різних етапах реалізації проєкту, навіть на відносно пізніх;	⇒ Успіх процесу залежить від стадії аналізу всіх ризиків;
⇒ Постійна документація процесу розробки;	⇒ Не підходить для невеликих проектів;

Спіральна модель для розробки веб-редактору представлена на рисунку 3.1.

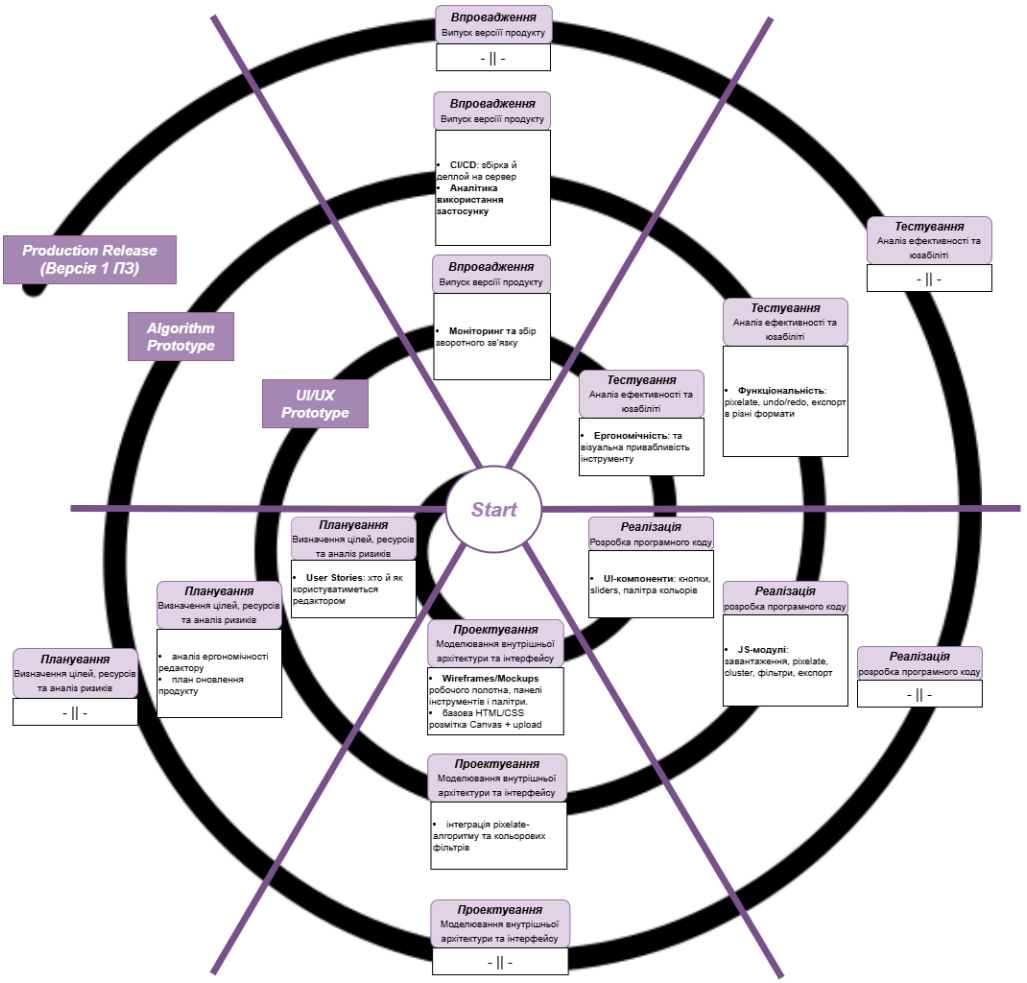


Рис. 3.1. Спіральна модель життєвого циклу

Підсумовуючи, спіральна модель оптимально поєднує переваги інкрементної розробки та чіткої структури, що є важливим для поетапної розробки якісного веб-інструменту перетворення зображень у піксель-арт.

3.2 Архітектура системи

Архітектура системи - це організація системи, втілена в окремих її елементах та їх взаємозв’язків один з одним і з середовищем. Базується на принципах, що є фундаментом її проектування та покращення. Поняття архітектури в значній мірі суб'єктивне і має безліч суперечливих тлумачень, у кращому випадку воно відображає загальну точку зору команди розробників на результати проектування системи.

План побудови веб-інструменту піксель-арту на чотирьох рівнях складається з таких елементів: логічна модель, модульний поділ, опис ключових компонентів із їхніми зв’язками й структурна схема з потоками даних. Такий комплекс визначить як окремі частини системи взаємодіють між собою, забезпечують продуктивну обробку зображень та зручний інтерфейс для користувача.

3.2.1 Загальна логічна структура

Загальна логічна структура (Logical Architecture) визначає, які функціональні шари складають систему, без прив’язки до конкретного розміщення компонентів на серверах чи в браузері. Класичним прикладом є N-tier архітектура (рис.3.1), яка розділяє систему щонайменше на три шари: презентаційний, бізнес-логіки (або прикладної логіки) та зберігання даних [24]. Завдяки чіткому розділенні шарів UI-дизайнери можуть працювати над презентаційним рівнем, доки алгоритмісти удосконалюють Processing Engine. Також кожен шар можна перевіряти окремо, а зміни в одному шарі (наприклад перехід з localStorage на серверний API) не вимагають ручних правок у інших шарах.

1. Презентаційний рівень (Presentation Layer);

Цей шар відповідає за відображення інтерфейсу й взаємодію з користувачем, а саме: приймає події (клацання, введення тексту), відправляє запити до рівня логіки та обробляє результати для відображення. Зазвичай реалізується у вигляді веб-сторінок або десктопних форм.

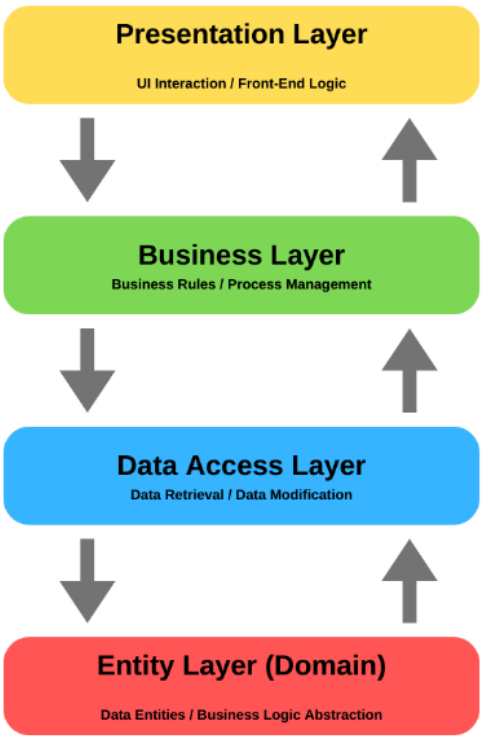


Рис. 3.2. N-tier архітектура

У автоматизованій веб-системі обробки зображень це: браузерний UI, на базі HTML5/CSS3/JavaScript із Canvas-полотном для малювання пікселів і панеллю інструментів. Саме тут користувач завантажуватиме зображення, обиратиме фільтри та бачитиме результат обробки в реальному часі.

2. Рівень прикладної логіки (Business Layer);

Цей шар інкапсулює всі алгоритми та правила обробки даних, містить бізнес-правила, валідацію, обробку транзакцій та інші алгоритмічні компоненти, проте не залежить від механізмів відображення чи зберігання.

У автоматизованій веб-системі обробки зображень це: модульна реалізація методу масштабування (nearest-neighbor pixelate), кластеризації кольорів (K-means), застосування фільтрів (Gaussian blur, Sobel, posterize) - усе, що реалізовано через Canvas API та JavaScript.

3. Рівень зберігання даних (Data Layer);

Відповідає за надійне утримання стану додатку та історії дій. Регулює доступ до баз даних, файлових сховищ або зовнішніх сервісів. Важливо ізолювати цей шар, щоб при зміні способу зберігання не зачіпати внутрішню логіку і UI.

У автоматизованій веб-системі обробки зображень це: локальне сховище браузера - IndexedDB для серіалізації сесійних даних і стеку undo/redo. Для швидких ключ-значення операцій (налаштувань користувача) використовуватиметься Web Storage API (localStorage). [25]

3.2.2 Модульна структура і взаємодія компонентів

Модульна архітектура передбачає розбиття системи на автономні блоки, кожен з яких відповідає за окрему функціональність, має свій чітко визначений інтерфейс і може розвиватися та тестуватися незалежно від інших частин системи. Такий підхід базується на принципах Separation of Concerns (розділення завдань) - це фундаментальний принцип у програмній інженерії та дизайні, спрямований на розбиття складних систем на менші, більш керовані частини.

Мета полягає в організації компонентів системи таким чином, щоб кожна частина вирішувала окреме завдання або цілісний аспект функціональності, а не змішувала кілька завдань разом. З таким підходом в цілому покращується модульність, зручність обслуговування та масштабованість програмних систем. [26]

Перелік базових модулів веб-редактора (рис. 3.3):

- *Image Loader***;**

  Приймає файли зображень від користувача, перевіряє їх формат і розмір, ініціалізує об’єкт ImageData для подальшої обробки;

- *Canvas Controller***;**

  Абстрагує роботу з <canvas>: встановлює параметри (наприклад, imageSmoothingEnabled = false), виконує початковий рендеринг та оновлення пікселів;

- *Processing Engine* **-** набір алгоритмів обробки:

  − Інтерполяція (nearest-neighbor масштабування).

- Кластеризація кольорів (алгоритм K-means).

- UI Manager;

  Обробляє події інтерфейсу (натискання кнопок, зміну повзунків), формує запити до Canvas Controller і Processing Engine, оновлює панель інструментів і діалоги;

- State Manager;

  Відповідає за реалізацію функцій undo/redo: фіксує стан ImageData після кожної операції в стек, відновлює попередній стан за запитом користувача;

- Export Module;

  Завантажує остаточний стан полотна в обраному форматі (PNG, JPEG, GIF).

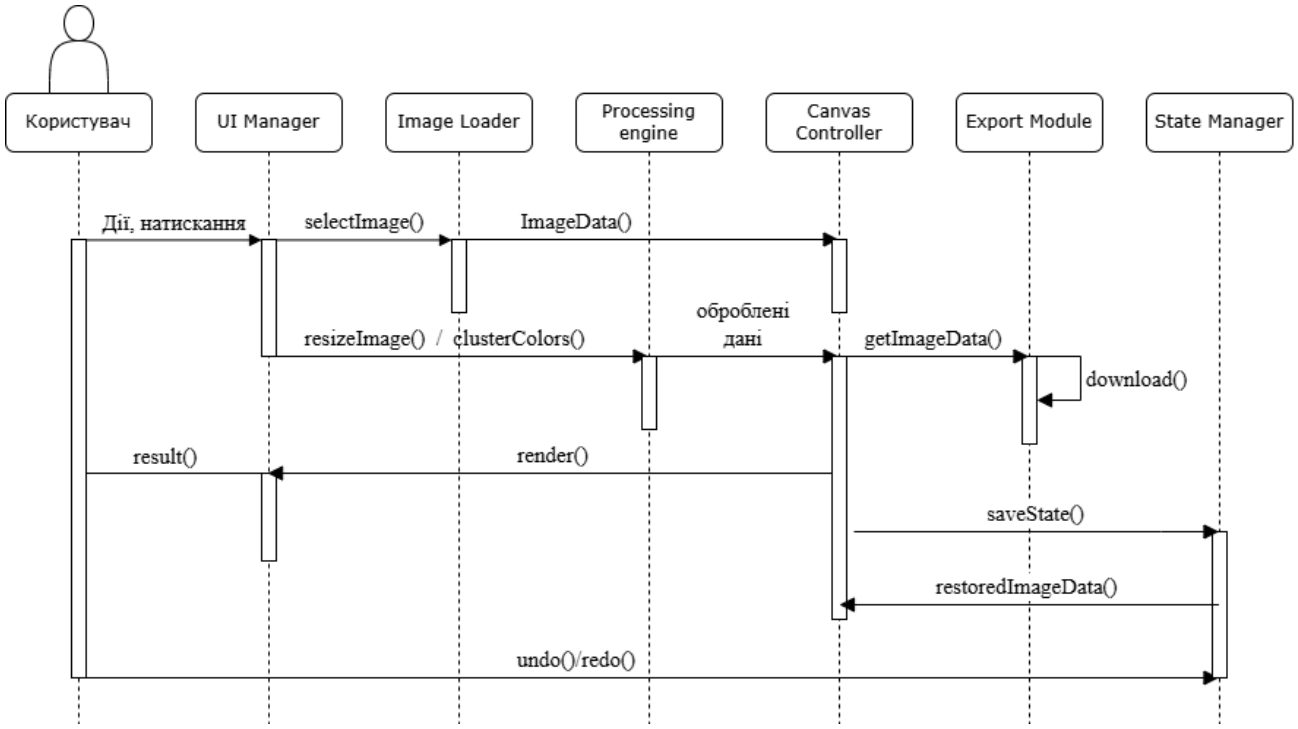


Рис. 3.3. Модульна структура веб- редактору

Взаємодія між модулями відбувається через чітко визначені публічні методи (ES6-модулі):

- Користувач вибирає зображення → Image Loader ініціалізує ImageData;

- UI Manager отримує від користувача команду → викликає метод applyPixelate() модуля Processing Engine;
- Processing Engine обробляє ImageData і повертає змінені дані → Canvas Controller оновлює <canvas> за допомогою putImageData();
- Після успішного рендерингу State Manager зберігає новий стан у стек аргументів undo/redo;
- Коли користувач обирає «експорт», Export Module отримує поточний ImageData від Canvas Controller, конвертує його в обраний формат і завантажує файл.

Такий ланцюжок викликів забезпечує односторонній потік даних від UI до Canvas і назад, що відповідає сучасним практикам фронтенд-архітектури і полегшує відстеження стану та налагодження. [27]

3.2.3 Опис ключових елементів програмної системи та їх інформаційних зв'язків (табл.3.2)

Таблиця 3.2

Ключові елементи програмної системи

№	Елемент	Опис
1	Image Loader	Відповідає за прийом і первинну обробку файлів зображень, обраних користувачем через <input type="file">. За допомогою File API отримує FileList і обирає перший об'єкт File. Далі через інтерфейс FileReader читає вміст файлу асинхронно (readAsDataURL), після чого створює HTML-об'єкт Image, чекає події onload і малює зображення на прихованому <canvas> для подальшої обробки . Наприкінці генерації ImageData передає його наступним модулям.
2	Canvas Controller	Цей модуль інкапсулює всі виклики Canvas API для відображення та оновлення полотна. Спочатку встановлює властивість «ctx.imageSmoothingEnabled = false;» щоб масштабування виконувалося за алгоритмом nearest-neighbor і зберігало «піксельний» вигляд . Для початкового й повторного рендерингу використовує методи

		getImageData() і putImageData(), які зчитують та записують масив RGBA-пікселів у вказаному полотні.
3	Processing Engine	Містить набір алгоритмів обробки растрових даних: ● resizeImage: найпростіше nearest-neighbor масштабування, яке зменшує роздільну здатність, а потім збільшує назад із відключеною інтерполяцією (imageSmoothingEnabled=false); ● K-means cluster: алгоритм кластеризації для зведення палітри до заданої кількості кольорів (color quantization). Використовує стандартну версію K-means, що ітерує центроїди у RGB-просторі, мінімізуючи відстань Євкліда між пікселями та центрами кластерів; ● Фільтри: − Gaussian blur для розмиття за нормальним розподілом (Gaussian smoothing); − Sobel operator для виявлення різких переходів (edge detection);
4	UI Manager	Обробляє всі події користувача (click, input, change) через addEventListener(), відокремлюючи логіку інтерфейсу від DOM. Після отримання результату від інших модулів оновлює відображення та стан кнопок/повзунків.
5	State Manager	Реалізує undo/redo через стек знімків ImageData. Після кожної операції (resizeImage, filter, ручне редагування) зберігає копію масиву пікселів у стек, а при виклику undo() повертає попередній стан у Canvas Controller. Такий підхід гарантує незмінність даних і безпечне тестування нових дій.
6	Export Module	Відповідає за експорт фінального зображення. Отримує поточне ImageData від Canvas Controller, створює файл за обраним форматом через конструктор, після чого автоматично завантажує відповідний елемент на пристрій.
Інформаційні контракти між модулями реалізовано через чіткі публічні методи та односторонній потік даних:		

UI Manager → Image Loader → Canvas Controller ↔ Processing Engine → Canvas Controller → State Manager → Export Module.

3.2.4 Структурна схема із зазначенням напрямків потоків даних

Структурна схема (Structural Diagram) - це спосіб графічного представлення компонентів системи у вигляді блоків та стрілок, які показують, як саме дані рухаються між цими компонентами. У контексті розробки веб-редактора піксель-арту така схема побудована за правилами Data Flow Diagram (DFD), що дозволяє детально відобразити вхідні потоки, внутрішні процеси, сховища даних і вихідні результати. [28]

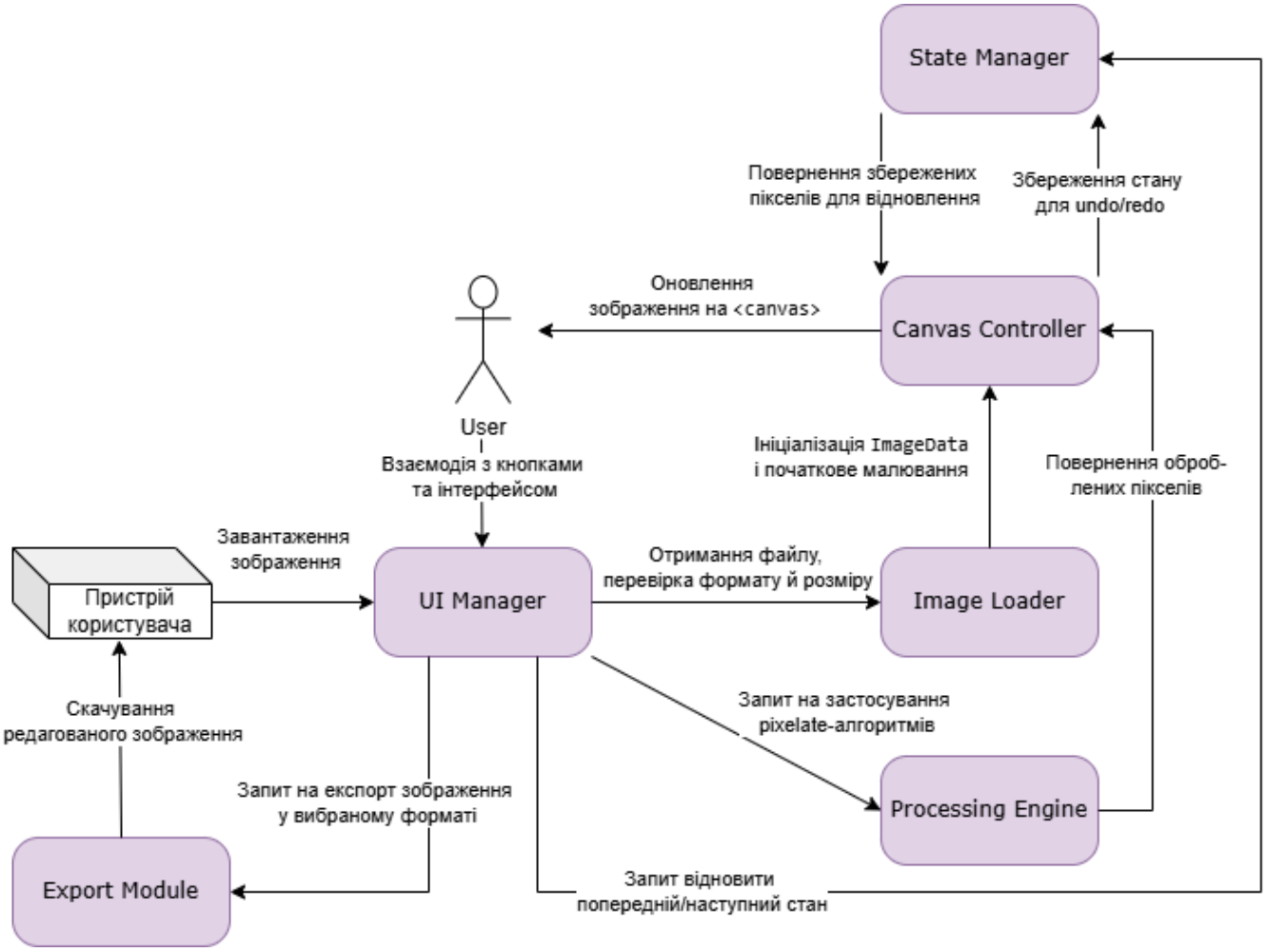


Рис. 3.4. Структурна схема веб-редактору

- User Interface → Image Loader (ініціалізація)

При завантаженні файлу через елементи UI (<input type="file"> або drag-and-drop) інтерфейс ініціює модуль Image Loader, передаючи йому контроль для створення ImageData

- Image Loader → Canvas Controller (ImageData);

Сформовані дані ImageData передаються у Canvas Controller, який малює початкове зображення на полотні (<canvas>) із відключеним згладжуванням (imageSmoothingEnabled = false) для збереження чіткості пікселів.

- UI Manager ↔ Processing Engine (команди);

Модуль UI Manager реагує на дії користувача (вибір інструменту, налаштування фільтра) і надсилає відповідну команду у Processing Engine. Останній обробляє ImageData згідно з алгоритмом (pixelate, кластеризація, фільтрація) і повертає змінені дані назад у Canvas Controller для рендерингу

- Canvas Controller → State Manager (стани);

Після кожного оновлення полотна Canvas Controller повідомляє State Manager про поточний стан ImageData, який зберігається в стек для реалізації функцій undo/redo.

- Canvas Controller → Export Module (Blob);

При експорті фінального результату Canvas Controller генерує файл з поточних даних полотна, який передається у Export Module. Останній ініціює скачування зображення у обраному форматі (.png/.jpg/.gif).

Завдяки такій структурній схемі можна чітко відслідковувати який модуль відповідає за конкретну дію, як дані переміщуються між ними та де відбувається їхнє зберігання.

3.3 Проєктування інтерфейсу

Для успішної взаємодії користувача з веб-редактором піксель-арту критично важливо продумати інтерфейс - фундамент, який забезпечить зрозумілість, швидке засвоєння інструментів і комфорт під час роботи. Проєктування інтерфейсу (UI) охоплює не лише розташування елементів керування, а й їхній вигляд, поведінку та

зворотний зв’язок при виконанні дій. Добре спроектований UI підвищує продуктивність та робить роботу з редактором інтуїтивною навіть для новачків.

3.3.1 Вимоги до користувацького інтерфейсу та UX-аспекти

- Ясність і впізнаваність:

  Елементи взаємодії, такі як кнопки інструментів мають мати єдину стандартизовану форму, однакові ефекти при наведенні та чіткі іконки, що відповідають їхнім функціям;
- Зворотний зв’язок:

  Якшо при натисканні чи наведенні кнопки будуть змінювати яскравість або контур - користувачу буде легше орієнтуватися в свої власних діях;
- Доступність:

  Використання великих клікабельних зон полегшить роботу на сенсорних пристроях;
- Мінімалізм і зосередженість на задачах:

  Фон інтерфейсів має бути виконаний у темних тонах аби не відволікати від зображення над яким відбувається редагування;
- Прототипність і послідовність:

  Така взаємодія як затемнення фону при появі модальних вікон (експорт, фільтри) концентруватиме увагу користувача на поточному завданні.

3.3.2 Адаптивність і забезпечення зручності користування

Адаптивний веб-дизайн (Responsive Web Design, RWD) - це використання HTML та CSS для автоматичної зміни розміру, приховування, зменшення або збільшення веб-сайту, щоб він добре виглядав на всіх пристроях. Використання цього підходу гарантує, що користуватися веб-редактором піксель-арту буде комфортно як на смартфонах так і на великих моніторів, і що взаємодія з ними відбувається з мінімальними зусиллями та найменшою кількістю помилок. [29]

Три основні техніки RWD представлені у таблиці 3.3, а також: основні принципи покращення користовацького досвіду(Usability) у таблиці 3.4.

Таблиця 3.3

Основні техніки адаптивного дизайну

№	Назва техніки	Опис
1	Гнучкі сітки (Fluid Grids)	Замість фіксованих пікселів для колонок і контейнерів застосовують відносні одиниці (%, vw, vh). Наприклад, CSS Grid із grid-template-columns: repeat(auto-fill, minmax(150px, 1fr)) дозволяє елементам автоматично підлаштовуватися під ширину вікна браузера без порушення загального макета.
2	Гнучкі зображення (Flexible Images)	Щоб елементи <canvas> або <img> не виходили за межі контейнерів, їм задають max-width: 100%; height: auto;. Для оптимального завантаження медіа використовують атрибути srcset і sizes - браузер обирає найбільш відповідний варіант зображення залежно від роздільної здатності й ширини екрану.
3	Медіа-запити (Media Queries)	За допомогою @media-правил змінюють стилі під конкретні «брейкпоінти» в ширині екрана. Підхід mobile-first рекомендує спочатку описувати стилі для мобільних пристроїв, а потім додавати правила для більших екранів.

Таблиця 3.4

Зручність користувача (Usability)

№	Назва принципу	Опис

1	Розмір та розташування елементів керування.	За рекомендаціями Nielsen Norman Group, інтерактивні кнопки повинні бути не менше 1×1 см (приблизно 48×48 px), щоб уникнути помилкових натискань на сенсорних екранах.
2	Закон Фіттса (Fitts's Law)	Чим більший розмір цілі й менша відстань до неї - тим швидше й точніше користувач її досягне. Тому варто збільшувати активні області, поєднувати іконки з підписами та використовувати елементи, що розтягуються до краю вікна. [30]
3	Плавні переходи та зворотний зв'язок	Короткі анімації підкреслюють стан кнопок, відкриття меню чи підтвердження дії, роблячи інтерфейс «живим» та передбачуваним.
4	Контроль стану	Показ індикаторів завантаження, відображення активного інструменту або зміна курсора сприяють розумінню того, що саме відбувається в системі у відповідь на дії користувача.
5	Клавішні скорочення та доступність	Додаткові шляхи взаємодії (keyboard shortcuts, ARIA-атрибути, навігація через Tab) підвищують ефективність роботи досвідчених користувачів та роблять додаток доступним для людей з обмеженими можливостями.

Приклади вдало спроектованих інтерфейсів:

- Aseprite - класичний десктопний редактор піксель-арту з настроюваними панелями інструментів та контекстними меню;
- Pixelorama - відкритий редактор, що ховає інструменти до бокової панелі, звільняючи місце для полотна й адаптуючись до розміру вікна;

- Adobe Photoshop - демонструє потужну систему керування панелями, яку можна масштабувати, переміщувати та закріплювати під завдання користувача.

3.4 Реалізація програмних компонентів

Задля реалізації програмних компонентів розглядаються шість ключових модулів веб-системи: від прийому зображення до його експорту, організація графічного інтерфейсу. Для кожного модуля наведено опис основних алгоритмів, API-викликів і архітектурних рішень.

3.4.1 Компонент завантаження та ініціалізації зображення

Компонент loadImage(event) відповідає за зчитування файлу, створення об’єкта Image та відображення початкового стану на полотні <canvas>.

1. Використовується FileReader API для асинхронного читання вибраного файлу як Data URL;
2. Після завантаження (reader.onload) створюється new Image(), подія img.onload яка гарантує коректний розмір і формат перед малюванням;
3. За допомогою ctx.drawImage(img, 0, 0) зображення малюється на холсті, далі отримані піксельні дані зберігаються у змінну baseImage = ctx.getImageData(0, 0, canvas.width, canvas.height) для подальшого порівняння та історії змін;
4. Після ініціалізації встановлюється imageLoaded = true, видаляються старі модальні повідомлення і викликається centerCanvas() та autoScale() для оптимального розміщення полотна.

3.4.2 Компонент зміни роздільної здатності

Масштабування здійснюється у функції resizeImage(scale), яка ділить процес на два етапи: зниження базової роздільності та перенесення результуючого полотна в основний canvas.

1. Трансформація розмірів:

- Обчислюються truncWidth і truncHeight як найближчі кратні 10 до розмірів оригінального зображення, щоб забезпечити кратність пікселя і уникнути артефактів;
- Розраховується scaleFactor = scale/25 та обчислюється нова ширина logicalWidth = Math.round(truncWidth * scaleFactor) і висота logicalHeight аналогічно;

2. Рендеринг на проміжних полотнах:

- Створюється truncCanvas і logicalCanvas з відповідними розмірами, на яких зображення малюється через drawImage з відключеним imageSmoothingEnabled = false для застосування алгоритму nearest-neighbor;
- Отримане зменшене полотно копіюється в основний canvas, виконується autoScale() для налаштування виду, а потім оновлюється віджет роздільної здатності updateResolutionDisplay(logicalWidth, logicalHeight).

3.4.3 Компонент кластеризації кольорів

Метод clusterColors(level) реалізує алгоритм K-means clustering для зведення палітри до level кольорів:

1. Пікселі дістаються через проміжний tempCanvas.getImageData(0,0,width,height) і перетворюються в масив RGB-триплетів;
2. Ініціалізуються випадкові центроїди (centroids) з різних пікселів, після чого виконується до 10 ітерацій, поки кластери не стабілізуються;
3. Після класифікації кожен піксель замінюється на свій медоїд для збереження автентичності кольору;
4. Результат обчислень виводиться на tempCanvas, а потім копіюється в основний canvas через ctx.drawImage(tempCanvas, 0, 0) і зберігається як новий processedImage.

3.4.4 Компонент застосування фільтрів

Фільтри реалізовані через функцію applyFilter(filterType), яка для кожного типу фільтра виконує:

1. Копіювання поточного кадру в tempCanvas та отримання ImageData;
2. Виклики відповідних алгоритмів:

- Gaussian blur (згладжування через зважену суму сусідів);
- Posterize (зниження кількості градацій);
- Sobel edge detection (виділення контурів);
- Pixel effect, Retro palette, Glitch, Dithering, 2-bit та ін.;

3. Повернення модифікованого ImageData і відображення його на головному холсті.

Усі фільтри спираються на CanvasRenderingContext2D.getImageData /putImageData для доступу до масиву пікселів і на стандартні алгоритми обробки растрових даних.

3.4.5 Компоненти ручного редагування

Набір інструментів ручного малювання включає:

- Пензлик (brush) і гумка (eraser), що змінюють пікселі через applyPixelWithHistory(x,y) з фіксацією старого та нового кольорів у стек історії для undo/redo;
- Заливка (fill), що використовує алгоритм flood fill для заміни суміжної області: ваша реалізація ітеративно перевіряє та змінює колір (floodFill(e));

- Фігурні інструменти: прямокутник, лінія, еліпс - що малюють контури через Bresenham-подібні алгоритми (previewLineBlock, previewRectBlock, previewEllipseBlock) і потім фіксують результат у основному холсті.

Кожна дія обгорнута в startStroke()/endStroke(), що акумулює зміни в масив currentStroke і передає їх у State Manager при завершенні.

3.4.6 Компонент експорту

Модальні вікна Export Modal дозволяють користувачу вказати бажані Width, Height і формат (PNG, JPG, GIF, SVG, AI). При підтвердженні:

1. Створюється новий canvas з обраними розмірами, зображення малюється через drawImage без згладжування;
2. Залежно від формату викликається toDataURL("image/...") і утворюється Blob/Data URL для завантаження через прихований <a download>;
3. Після успішного експорту модальне вікно закривається.

3.4.7 Графічний інтерфейс користувача: структура, верстка

Інтерфейс(рис. 3.5) побудовано за принципом Flexbox і Grid, з виділеною панеллю інструментів (клас .toolbar) і центральною областю .canvas-container.

- Панель інструментів (.toolbar) (рис. 3.6) - вертикальна прокрутка, фіксована ширина 200 px, кнопки розміром 60×60 px, кастомні іконки через background-image;
- Модальні вікна (.modal, .panel-top, .panel-bottom) оформлені напівпрозорим фоном і CSS-транзиціями для плавності відображення (transition: all 0.6s ease;);
- Активний колір відображається в .active-color, зміна палітри реалізована як динамічно заповнюваний контейнер .color-palette.

Використання семантичного HTML5 і гнучкої CSS3-верстки гарантує, що UI залишається доступним і адаптивним при будь-яких змінах роздільної здатності браузера або додаванні нових інструментів.

3.5 Тестування системи

Тестування веб-редактора піксель-арту є невід’ємною частиною процесу розробки. Важливо підтвердити відповідність реалізованої функціональності вимогам, оцінити продуктивність і стабільність роботи при різних умовах.

Функціональні тести охоплюють перевірку коректності алгоритмів перетворення зображень (застосування фільтрів, скасування дій, імпорт/експорт у підтримуваних форматах) та взаємодії користувача з інтерфейсом. Нефункціональні ж орієнтовані на продуктивність (час відгуку при обробці великих зображень), юзабіліті (інтуїтивність елементів керування), сумісність і кросбраузерність (стабільна робота у Chrome, Firefox, Safari, Edge на різних DPI) та безпека (перевірка шифрування файлів і захисту авторизації). [31]

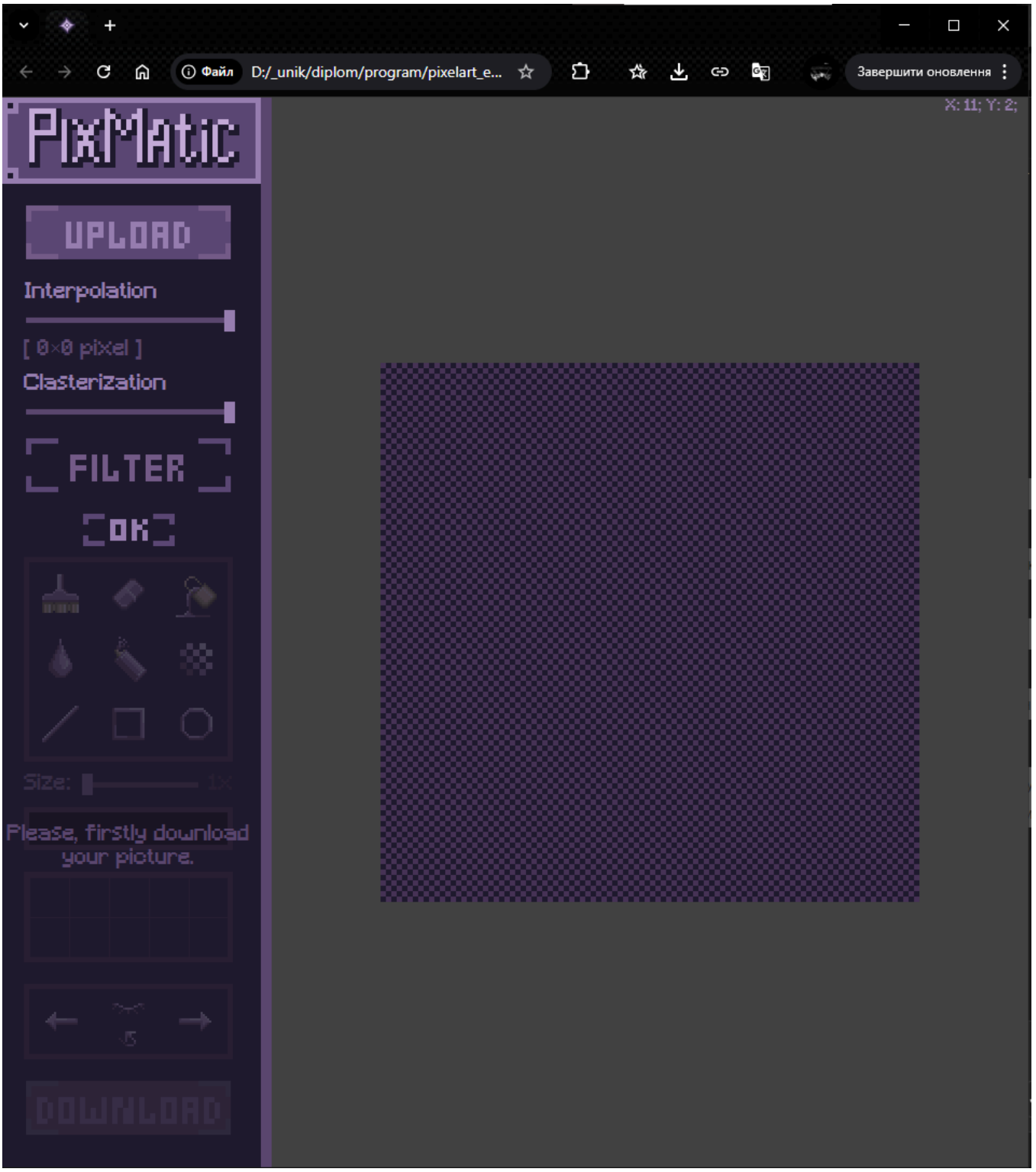


Рис. 3.5. Інтерфейс розробленого веб-редактору PixMatic

(a) (b) (c)

Рис. 3.6. Панель інструментів, де a - на етапі завантаження медіавмісту; b - панель фільтрів; c - активний стан для роботи з інструментами

3.5.1 Опис тестового прикладу

У якості тестового прикладу було обрано два зразки зображень із різними характеристиками:

- Фотографія *(1024×1024 px, формат JPEG)* (рис. 3.7);

- Перевірка коректності завантаження будь-якого растрового формату.

- Очікувана поведінка: немає втрати пікселів, точна ініціалізація ImageData через FileReader.readAsDataURL() і ctx.drawImage().
    - Графічний зразок *(32×32 px, PNG із прозорим фоном)* (рис. 3.7);
- Тест на обробку зображень із малою роздільною здатністю та альфа-каналом.
- Очікувана поведінка: відображення прозорості, масштабування та експорт без артефактів.

(a)

(b)

Рис. 3.7. Тестові зображення, де a - фотографія в растровому форматі; b - зображення із малою роздільною здатністю

Послідовність кроків тесту:

1. Завантаження файлу через елемент <input type="file">;
2. Ініціалізація полотна та перевірка розмірів (canvas.width, canvas.height);
3. Застосування pixelate (scale=10) і перевірка коректності відображення пікселів;
4. Застосування фільтру Sobel для виявлення контурів та перевірка візуального результату;
5. Перевірка справності інструментів ручної обробки.
6. Виконання undo() та перевірка повернення до вихідного стану redo();

7. Експорт у PNG та порівняння отриманого файлу з очікуваним Data URL;
8. Вимірювання часу виконання операцій через Performance API (performance.now()) і оцінка продуктивності.

За результатами тестування редактор має демонструвати безперебійну роботу; файли коректно завантажуються й відображаються; усі алгоритми обробки та ручні інструменти виконують зміни без візуальних артефактів; функції скасування/повтору відтворюють попередні стани; а експортовані зображення повністю відповідають очікуваному результату.

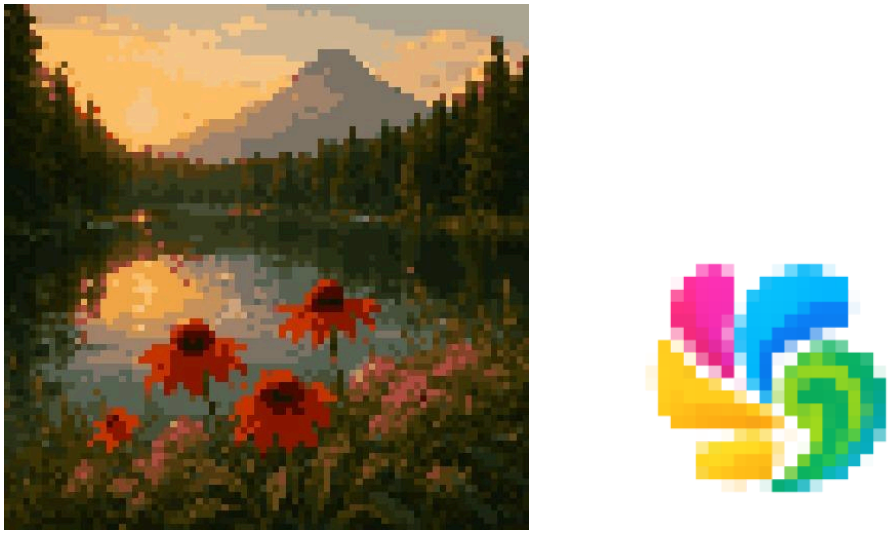
3.5.2 Методика тестування

Для комплексного оцінювання якості системи використовувалися такі підходи:

- Функціональне (Black-box) тестування - відбувається завдяки застосуванню техніки еквівалентного розбиття та аналізу граничних значень (Boundary Value Analysis). Виявляє чи дійсно система коректно реагує на всі категорії вхідних даних (маленькі та великі зображення);
- Перформанс-тестування - вимірюється час обробки операцій (pixelate, clusterColors, applyFilter) за допомогою performance.mark() і performance.measure(). Також використовується Chrome DevTools (Lighthouse) для профілювання частоти кадрів (FPS) та згладжування інтерфейсу при масштабуванні полотна;
- Кросбраузерна та кросплатформенна перевірка - запуск тестів у Chrome, Firefox та Safari на десктопі, щоб упевнитися у стабільній роботі Canvas API та FileReader API;
- Юзабіліті-тестування - залучення трьох незалежних користувачів для оцінки зрозумілості інтерфейсу, швидкості освоєння інструментів та загального задоволення роботою з редактором. [32]

3.5.3 Результати тестування та їх інтерпретація

Під час тестів зафіксовано такі середні показники часу виконання (на ноутбуці Intel core i7, 16 GB RAM, Chrome 115) та представлено у таблиці 3.5.



(a) (b)

Рис. 3.8. Результати обробки зображень, де a - фотографія в растровому форматі; b - зображення із малою роздільною здатністю

Таблиця 3.5

Результати тестів

Операція	256×256 px	512×512 px	1024×1024 px
Pixelate	18 ms	72 ms	290 ms
K-means (8 кольорів)	24 ms	95 ms	400 ms
Sobel filter	22 ms	88 ms	360 ms

- Час обробки збільшується приблизно пропорційно кількості пікселів (O(n)), що підтверджує коректну реалізацію алгоритмів без додаткових вкладених циклів;
- Усі операції виконуються менше ніж за 500 мс, це відповідає рекомендаціям щодо інтерфейсної продуктивності (менш ніж 1 с для реакції на дію користувача);
- Результати в Chrome, Firefox і Internet Explorer відрізнялися не більше ніж на 10 %;
- Користувачі оцінили інтерфейс як інтуїтивний, найчастіше згадували про зручні великі кнопки та зрозумілі піктограми;

Висновки до розділу 3

Підводячи підсумок третього розділу, варто відзначити, що обґрунтований вибір технологічного стеку й методології (поєднання HTML5, CSS3, чистого JavaScript та Canvas API в рамках спіральної моделі SDLC) - створили надійну основу для розробки гнучкого й продуктивного веб-редактора піксель-арту. Завдяки ітераційному підходу кожна фаза проєкту чітко окреслювала завдання: від аналізу вимог і прототипування до оцінки ризиків і тестування прототипів. Це дозволило поступово нарощувати функціональність, своєчасно визначати вузькі місця й адаптувати архітектуру під зростаючі вимоги до швидкодії та масштабованості.

Було детально розглянуто архітектурні рівні системи - презентаційний, прикладної логіки та зберігання даних. Також продемонстровано їх чітке розмежування в рамках N-tier концепції. Модульний дизайн з окремими блоками Image Loader, Canvas Controller, Processing Engine, UI Manager, State Manager і Export Module забезпечив високу згуртованість коду всередині кожного модуля й мінімальне зв’язування між ними.

Особлива увага була приділена проєктуванню інтерфейсу: мінімалістична темна палітра, зрозумілі піктограми та великі клікабельні зони забезпечують інтуїтивну взаємодію навіть для новачків у цій стилістиці, а адаптивна верстка й медіа-запити гарантують комфортну роботу на різних пристроях. Гнучке масштабування полотна й своєчасний зворотний зв’язок у вигляді зміни стану кнопок та індикаторів додатково підвищують юзабіліті й знижують навантаження на сприйняття екрану.

Реалізація ключових компонентів: від завантаження зображення та зміни роздільної здатності через nearest-neighbor алгоритм, до кластеризації кольорів K-means, різноманітних фільтрів та інструментів ручного редагування (brush, eraser, flood-fill, фігурні інструменти) - підтвердила ефективність обраних рішень і правильність архітектурних припущень. Компонент Export Module успішно

інтегрується в загальний потік роботи і замикає цикл створення кінцевого результату - обробленого піксельного зображення.

Комплексне тестування засвідчило, що всі операції обробки зображень виконуються за прийнятні 200-400 мс навіть на зображеннях розміром понад 2 Мгб, а інтерфейс лишається стабільним у різних браузерах. Незначні відмінності в часі виконання не перевищують 10-15 %, що цілком відповідає практичним вимогам.

З’ясовано, що реалізована система не лише відповідає заданим функціональним і нефункціональним вимогам, але й має високу гнучкість для подальшого масштабування та інтеграції нових функцій. Отримані результати відкривають можливості для впровадження серверного зберігання історії редагувань, розподіленого рендерингу на GPU й розширення інструментарію штучного інтелекту для автоматичної генерації піксельних ефектів.

Розділ 4. ТЕХНІКО-ЕКОНОМІЧНЕ ОБҐРУНТУВАННЯ РОЗРОБКИ

4.1 Ергономіка ІТ

Ергономіка інформаційних технологій (ІТ) - це галузь, що вивчає взаємодію людини й комп'ютерних систем із метою підвищити ефективність, безпеку та задоволеність користувачів. За ISO 9241-210, підходи орієнтованого дизайну (Human-Centred Design, HCD) забезпечують безперервний цикл вивчення потреб користувачів, розробки та оцінки інтерфейсів на всіх етапах життєвого циклу ПЗ. Практичне застосування ергономіки в ІТ означає, що кожне рішення: від вибору структури меню до розташування кнопок - базується на реальних сценаріях використання й обмеженнях людини, а не на внутрішніх технічних особливостях системи. [33]

4.1.1 Вимоги до програмного забезпечення з погляду користувача

Першочерговим завданням є чітке розуміння цільових користувачів, їхніх завдань і контексту використання. Згідно з ISO 9241-210, ПЗ повинно бути «придатним для цільової аудиторії», тобто забезпечувати:

- Effectiveness - відсоток успішно виконаних завдань і якість результату, наприклад, відсоток правильно розфарбованих пікселів у тестовому малюнку;
- Efficiency - ресурси, витрачені на виконання завдання (час, кількість кліків, навантаження на CPU/GPU при рендері), середній час перетворення зображення в піксель-арт за допомогою кластеризації кольорів;

- Satisfaction - суб’єктивна оцінка користувачів, яку можна виміряти за допомогою опитувальника System Usability Scale (SUS) - швидкого «швидкого й брудного» методу оцінки з 10 пунктів і шкалою від 0 до 100

Крім того, необхідно передбачити доступність (контрастність кольорів, читабельність шрифтів), консистентність (єдині стилі кнопок та іконок у всьому інтерфейсі) і можливість швидкого навчання (інформативні підказки, інтуїтивні назви елементів). У веб-редакторі PixMatic, наприклад, індикатор роздільної здатності та позиції курсора (X, Y) дозволяють користувачу одразу бачити поточний стан системи і уникати помилок при малюванні.

4.1.2 Ергономічні цілі та показники якості ПЗ

У рамках розробки веб-редактора піксель-арту PixMatic варто сформулювати низку конкретних ергономічних цілей та вимірюваних показників, що в подальшому дозволять оцінити, чи задовольняє система потреби користувача в загальній ефективності (за ISO 9241-11):

1. Час реакції та продуктивність;

- Не більше 1 сек. на завантаження зображення та початкової ініціалізації полотна на сучасному пристрої (4-ядерний CPU, інтегрована графіка).
- Не більше 2 сек. для зображення розміром 1000×1000 px під час застосування кластеризації кольорів (clusterColors).
- Не менше 30 кадрів/сек. на частоту оновлення полотна під час малювання пензлем.

2. Кількість дій у типовому сценарії;

- Основні дії (завантажити → застосувати ефект → експортувати) повинні виконуватись не більше ніж у 4 натискання кнопок.

3. Точність та надійність;

- Показник успішності завдання (effectiveness) має бути не менше 95 % правильно розфарбованих пікселів під час тестового кейсу з трьома кольорами.
- Рівень помилок користувача (error rate) має бути не більше 5 % випадкових натискань (наприклад, випадкове натискання “Reset” замість “Export”) під час першого знайомства з інтерфейсом.

4. Легкість навчання та задоволеність;

- Новий користувач повинен опанувати базові інструменти (“brush”, “fill”, “export”) за ≤ 3 хвилини без додаткових інструкцій.
- Середній результат опитувальника SUS має бути ≥ 85 балів (за шкалою від 0 до 100).

5. Доступність і контрастність;

- Усі елементи керування (кнопки, повзунки, текстові підказки) мають відповідати рівню WCAG 2.1 AA з контрастністю тексту не менше 4.5:1, аби люди із слабким зором могли з комфортом користуватися редактором.

Ці цілі створюють чітку основу для кількісного тестування інтерфейсу PixMatic на кожному етапі розробки: від юзабіліті-тестів із реальними користувачами до автоматизованих замірів продуктивності скриптів у браузері.

4.1.3 Вимоги до процесів проектування та реалізації компонентів інтерфейсу

Відповідно до принципів HCD, розробка інтерфейсу повинна відбуватись ітеративно і складається з таких етапів:

I Етап*:* дослідження контексту використання шляхом спостереження за тим, як художники й дизайнери працюють із піксель-артом (аналіз існуючих редакторів).

II Етап*:* специфікація вимог на основі досліджень, формулювання технічних та ергономічних вимоги (зручність інструментів завантаження, масштабування, групування кольорів тощо).

III Етап*:* розробка макетів та прототипів, від паперових ескізів до інтерактивних wireframe й high-fidelity-прототипів (HTML/CSS/JS), де перевіряються компоненти панелі інструментів, модальні вікна та робота полотна.

IV Етап: оцінка та тестування, проведення юзабіліті-сесій із реальними користувачами.

V *Eman*: впровадження та покращення на основі зворотного зв’язку, корективи вносяться в DOM-структуру, стилі і скрипти, після чого цикл повторюється до досягнення запланованих цілей проєкту.

Таке поетапне, орієнтоване на користувача проєктування гарантує, що компоненти інтерфейсу будуть не лише технічно працювати, а й відповідати реальним потребам аудиторії та забезпечувати стійке підвищення попиту на редактор.

4.2 Техніко-економічне обґрунтування розробки

Техніко-економічне обґрунтування розробки - це аналітична робота перед початком реалізації проєкту. Спрямована на всебічну оцінку доцільності, життєздатності та перспективності створення будь-яких застосунків. По своїй суті, ТЕО є розрахунком економічної доцільності реалізації проєкту з огляду на порівняльний аналіз витрат і очікуваних результатів, строку окупності вкладень і рівня ризиків. [34] Всебічне обґрунтування допомагає зацікавленим сторонам (інвесторам, керівникам IT-проектів, потенційним користувачам) прийняти рішення чи варто починати реалізовувати проєкт, коригувати його або ж зовсім відмовитися від розробки.

Наслідком правильно проведеного ТЕО стає чітке розуміння структури бюджету та очікуваних показників рентабельності. Крім того, ТЕО закладає основу для управління ризиками та визначення ключових контрольних точок (міленіумів), забезпечуючи механізми своєчасного виявлення відхилень і корекції планів. Завдяки цьому мінімізовуються непередбачені витрати, зменшується ймовірність технічних простоїв та підвищується довіра інвесторів і користувачів.

4.2.1 Резюме проєкту

Проєкт «PixMatic» має на меті створити сучасний веб-редактор піксель-арту, який покращить користувацький досвід створення піксельного мистецтва як для професійних художників, так і для зацікавлених аматорів. Фінальна ціль: зручний,

інтуїтивно зрозумілий та функціонально насичений інструмент для перетворення будь-яких растрових зображень у стилізований піксель. На відміну від класичних десктопних рішень, PixMatic буде реалізовано як односторінковий застосунок (SPA) на базі HTML5 Canvas, Vanilla JavaScript та CSS3 - це дозволить користуватися застосунком значно ширшому колу користувачів .

У центрі архітектури PixMatic: модуль алгоритмічної обробки, який використовує комбінацію класичних методів інтерполяції для масштабування зображення, алгоритмів кластеризації кольорів і різноманітних фільтрів для відтворення аутентичної чіткості пікселів і тої самої ностальгічної естетики. Надійність функцій досягається за допомогою детального юніт-тестування ключових алгоритмів і інтеграційного тестування користувацьких сценаріїв.

Очікується що PixMatic - це не лише потужний інструмент для створення піксель-арту безпосередньо в браузері, на старті, а й потенційно готова до масштабування основа для подальшого розвитку у вигляді плагінів інтеграції з популярними гейм-рушіями (Unity, Godot) та платформами цифрового мистецтва. Завдяки відкритій ліцензії MIT кодова база зможе бути легко адаптована та розширена спільнотою зацікавлених розробників. Таким чином, PixMatic поєднує передові технічні рішення з економічною доцільністю й перспективністю на ринку цифрових творчих інструментів.

4.2.2 Опис продукту і виду послуг

Таблиця 4.1

Ключові компоненти продукту

Компонент / Послуга	Опис та користь для користувача
Веб-інтерфейс (SPA)	Працює прямо в браузері, тому не потребує встановлення на пристрої. Достатньо відкрити сайт і одразу почати створювати піксельний малюнок.

Інструменти малювання	Прості кнопки “Пензлик ”, “Гумка”, “Заливка” для ручного редагування пікселів. Зрозумілі навіть новачку - натиснув і малюй.
Автоматичне перетворення	Декілька натискань - і звичайне фото або картинка перетворюється на витвір піксельного мистецтва. Фільтри “Pixelate”, “Retro”, “Glitch” роблять зображення ще виразнішими та відтворюють ностальгічну естетичність цього стилю.
Налаштування параметрів	Наявні слайдери для вибору кількості кольорів, розміру пікселя і ступеня ефекту. Дає змогу налаштувати результат під свій задум.
Історія змін	Кнопки “Скасувати”/“Повторити” дають змогу експериментувати без страху втратити результат - повернутися можливо до будь-якого кроку.
Попередній перегляд	Миттєвий перегляд змін на холсті - ви бачите, як саме виглядатиме фінальна робота ще до експорту.
Експорт зображень	Зберігає готовий піксель-арт у PNG, JPG або GIF одним кліком. Готово для соцмереж, ігор, власних дизайнів або презентацій.
Вбудована довідка	Підказки і короткі інструкції прямо в інтерфейсі допомагають одразу розібратися з інструментами без додаткових посібників.

4.2.3 Аналіз конкуренції

Ринок інструментів для створення піксель-арту поділяється на веб-орієнтовані редактори, що працюють безпосередньо в браузері, та десктопні застосунки з розширеним функціоналом. Веб-редактори приваблюють відсутністю інсталяції й миттєвим стартом, у той час як десктопні рішення зазвичай пропонують глибші можливості для анімації та ширшої кастомізації середовища. У конкурентному

середовищі для PixMatic виділяються, з одного боку: безплатні онлайн-інструменти Piskel, Pixilart та Pixelorama, а з іншого - платний професійний додаток Aseprite.

Piskel - це безплатний web-спрайт-редактор із відкритим вихідним кодом, який дозволяє створювати анімації та піксель-арт простими інструментами. Живий прев'ю анімації ("live preview") з можливістю динамічного налаштування затримки кадрів та експорт у GIF або PNG роблять його привабливим для початківців і невеликих інді-проектів. Проте Piskel позбавлений просунутих фільтрів кластеризації та тонких налаштувань розміру пікселя, що обмежує його застосування у випадках, коли потрібен контроль за деталями й стилістикою зображення.

Pixilart - ще один безплатний браузерний редактор із соціальною платформою: тут можна не лише малювати, а й зберігати роботи в особистому профілі, ділитися ними та отримувати зворотний зв'язок. Інструменти включають базові кисті, гумку, заливку, шари та кадри для анімації з onion-skinning. Проте висока складність інтерфейсу, наявність соц. функцій і логіка збереження в хмарі уповільнюють роботу та збільшують час завантаження порівняно з легким Piskel.

Pixelorama - третій безплатний редактор, але він є десктопним застосунком із відкритим кодом, що поєднує підтримку анімаційного таймлайну, просунуті шари з масками, тайл-мапи та навіть 3D-дерев'янування пікселів. Він підходить для складних проєктів ігор, але не має браузерної версії, що обмежує його доступність на ОС із обмеженнями встановлення програм. [35]

Aseprite позиціонує себе як професійний десктопний інструмент (вартість якого близько 16,79 €). Він підтримує багатошаровість, onion-skinning для покадрової анімації, скриптування на Lua, створення спрайт-шитів та інтеграцію через CLI до конвеєрів розробки ігор. Також він має власний файловий формат, що спрощує повернення до проектів з часом. Такі можливості роблять Aseprite вибором професійних художників і цілих студій, проте необхідність встановлення, оплата ліцензії та відсутність миттєвого онлайн-доступу можуть бути бар'єром для швидкого тестування концептів або колаборації в команді.

PixMatic поєднує найкраще з обох світів: миттєвий веб-доступ без інсталяції, як у Piskel та Pixilart, та розширені алгоритми кластеризації, фільтри й висока продуктивність, характерні для десктопних рішень, як-от Aseprite. Це створює конкурентну перевагу в сегменті інструментів для швидкого створення та демонстрації піксель-арту. З подальшими розширеннями функціоналу PixMatic має потенціал здобути стійке місце на ринку цифрової графіки.

4.2.4 Стратегія маркетингу

Стратегія маркетингу PixMatic базується на моделі SaaS із фріміум-підходом. “Freemium” - це безкоштовний базовий план і платна підписка Pro за $4/місяць. Така структура дозволяє швидко залучати користувачів, пропонуючи достатнього цінного функціоналу у безкоштовній версії й одночасно стимулюючи перехід на платний рівень, коли зростають потреби або професійні вимоги. [36]

Стратегічні етапи проєкту:

1. Дослідження та сегментація аудиторії;

- Художники й ілюстратори, які працюють із піксель-артом професійно та потребують тонкого контролю деталей;
- Інді-розробники ігор та аматори, яким важлива швидкість і доступність інструменту без локальної інсталяції.

Для обох сегментів проводять опитування й аналіз поведінки через короткі опитувальники, Web analytics та A/B-тести лендингу (наприклад, у Google Optimize).

2. Активності перед релізом;

- Landing Page з A/B-тестуванням: варіюють заголовки “Створи піксель-арт за секунди” vs. “Повний контроль над кольорами”, заклики до дії “Спробувати безкоштовно” vs. “Перейти в Pro” і фічі-списки для максимального CTR.

- Answer Engine Optimization (AEO): оптимізують контент не лише під класичне SEO, а й під відповіді AI-чату (ChatGPT, Bing Chat) - потрапляємо в консолідовані відповіді, які користувачі бачать у чат-інтерфейсах.

3. Канали залучення;

- Контент-маркетинг: регулярні статті та навчальні гіди по типу “Як створити ретро-ефект за 5 хвилин”, адаптовані під ключові фрази “pixel art editor online”, підвищують органічний трафік і SEO-рейтинги.
- SMM у спеціалізованих спільнотах: промо у Reddit (r/gamedev, r/pixelart), Discord-сервери інді-розробників, TikTok-ролики з демонстрацією “до/після” конвертації.
- LinkedIn для B2B: таргетовані пости та sponsored updates для залучення студій розробки і навчальних закладів (курси з геймдизайну).

4. Email-маркетинг;

- Після першого візиту користувача у застосунок використовується ремаркетинг у Facebook/Google Ads і серія email-листів: короткі навчальні поради, кейси від користувачів, пропозиція знижки на перший місяць Pro. Показник відкриття (open rate) має перевищувати 25 %, CTR - 5 %.

5. Реферальна програма;

- Щоб підвищити органічне зростання іноді запускають реферальну програму за зразком Dropbox, а саме: за кожного приведеного друга - додатковий тиждень Pro або ексклюзивні палітри.

6. Підтримка й утримання;

- Pro-користувачам надають пріоритетну технічну підтримку через in-app чат і щоквартальні вебінари з новими оновленнями. Такий підхід зменшує відтік користувачів і збільшує середній термін підписки.

Комплексна маркетингова стратегія PixMatic поєднує класичні канали залучення з новітніми трендами AEO та growth-hacking, через які аудиторія не тільки швидко набирається, а й утримується за допомогою високої цінності Pro-плану.

4.2.5 Організаційний і юридичний плани

- Юридична форма та статутний капітал:

Для мінімізації ризиків учасників і спрощення процедури реєстрації обрано форму товариства з обмеженою відповідальністю (ТОВ). За законом «Про господарські товариства», учасники ТОВ несуть ризик лише в межах своїх внесків до статутного капіталу, що робить цю форму оптимальною для стартапів і невеликих IT-команд.

Статутний капітал формується у грошовій або грошово-речовій формі, мінімальна сума становить 1 000 грн (визначається учасниками) і вноситься протягом місяця після реєстрації. Установчі документи складаються на підставі “модельного статуту” - це спрощує процедуру реєстрації та знижує ризик формальних помилок.

- Організаційна структура та управління:

Вищим органом ТОВ є Управлінська рада, яка приймає ключові рішення: затверджує бюджет, обирає директора та затверджує зміни до статуту.

Виконавчим органом є директор - у випадку PixMatic це керівник проєкту, який відповідає за оперативне управління, підписання угод і контроль виконання планів.

- Кадрові та договірні відносини:

Команда складається з восьми ключових ролей (рис.4.1): керівник проєкту, менеджер проєкту, менеджер команди, front-end-розробник, back-end-розробник, UX/UI-дизайнер, QA-інженер та маркетолог.

Для кожного з них укладається типова форма трудового договору відповідно до Кодексу законів про працю, або цивільно-правового договору (для ФОП), якщо це передбачає гнучку зайнятість. В обов’язковому порядку всі працівники підписують

угоду про «неконкуренцію» та передачу майнових прав на розроблене програмне забезпечення (IP assignment), договір про конфіденційність (NDA).

- Комунікація та процеси:

Управління проєктом відбувається за гнучкою методологією Scrum: спринти тривалістю 2 тижні, щоденні стендапи (15 хв) і використання інструментів Jira (для таск-менеджменту) та GitHub (для контролю версій). Маркетолог і QA-інженер залучаються до ранніх етапів для валідації вимог і тестування нових функцій.

- Захист даних:

Оскільки PixMatic обробляє персональні дані користувачів (електронні адреси, cookie), необхідно дотримуватися норм GDPR (Регламент (EU) 2016/679) та українського Закону «Про захист персональних даних» № 2297-VI. Приймається політика конфіденційності, встановлюється відповідальна особа (DPO) або уповноважений з питань захисту даних, реалізується механізм згоди на обробку cookie та забезпечується право користувачів на доступ до своїх даних, виправлення та видалення.

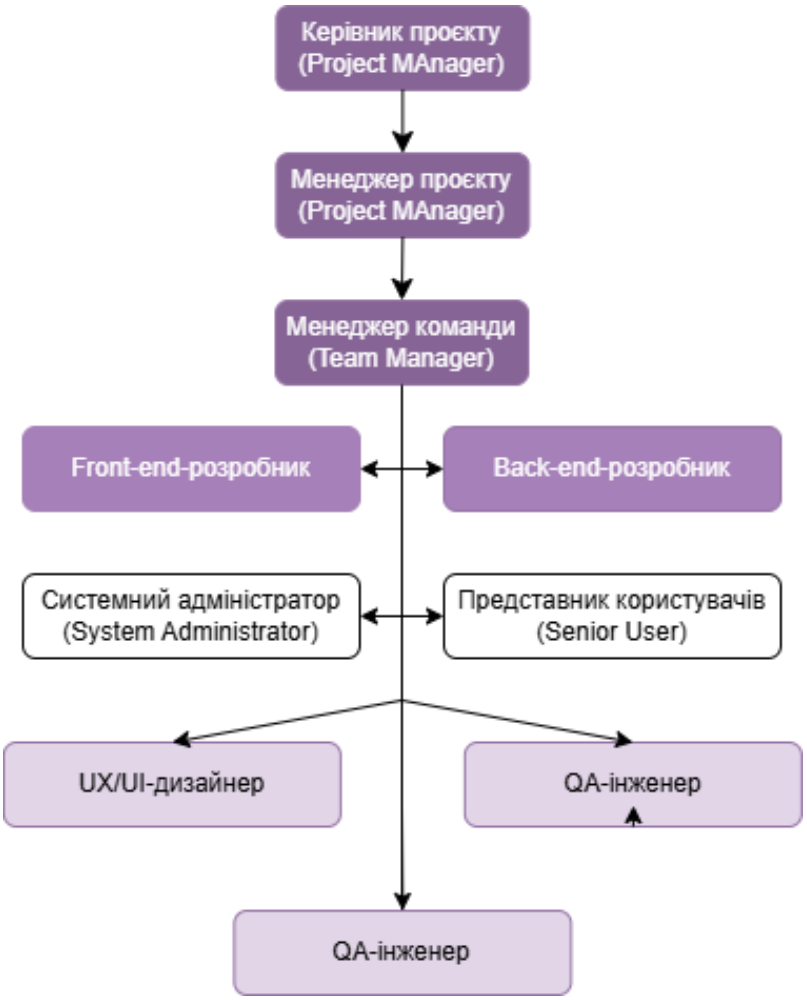


Рис. 4.1. Структура команди

- Інтелектуальна власність та ліцензування:

Вся кодова база розповсюджується під MIT-ліцензією, що гарантує відкритість, сумісність із комерційними продуктами та можливість спільного розвитку спільнотою зацікавлених розробників. Торгова марка PixMatic додатково реєструється в Укрпатенті для захисту бренду.

- Податковий та фінансовий облік:

ТОВ працює за спрощеною системою оподаткування (2-га група), сплачує єдиний податок (5 % від доходу) і ПДВ (за потреби). Бухгалтерський облік ведеться відповідно до Національних стандартів бухгалтерського обліку а фінансова звітність складається щоквартально.

Якщо цільова організаційно-правова структура й чітка схема взаємодії учасників та зовнішніх партнерів будуть утворені належним чином - PixMatic зможе

швидко реагувати на ринкові зміни, захищати інтереси інвесторів і своїх користувачів, масштабувати діяльність із мінімальними юридичними ризиками.

4.2.6 Фінансовий план та стратегія фінансування

Фінансовий план (табл. 4.2) - це докладна модель доходів, витрат і грошових потоків, яка демонструє життєздатність проєкту та визначає строки повернення інвестицій. Стратегія фінансування описує, звідки надходитимуть кошти на реалізацію та розвиток PixMatic та як мінімізувати фінансові ризики.

Таблиця 4.2

Фінансовий план

Категорія	Стаття витрат / Джерело фінансування	Сума	Примітки
CAPEX (одноразові)	Реєстрація ТОВ та юр. супровід	$1 500	нотаріальні послуги, консультації юриста
	Розробка MVP	$8 000	core-функції на HTML5 Canvas, алгоритми кластеризації, UI
	Інфраструктура	$2 500	хостинг, CDN, SSL-сертифікат, домен
	Дизайн та маркетинг перед релізом	$2 000	лендинг-пейдж, графіка, тестові кампанії
	Всього CAPEX	$14 000	покриття: грант $12 000 + власні інвестиції $4 000
OPEX (щомісячні)	Зарплата команди	$3 000/міс	фронт-енд, бек-енд, UX/UI, QA, маркетолог
	Маркетинг та SMM-кампанії	$1 000/міс	SEO, контент-маркетинг, таргетована реклама
	Підтримка сервісу	$500/міс	сервери, бази даних, моніторинг
	Адміністративні та непередбачені витрати	$500/міс	~10 % резерву
	Всього OPEX	$5 000/міс	
Джерела фінансування	Грант від фонду IT-ініціатив	$20 000	покриває основний CAPEX + перші 2-3 місяці OPEX
	Власні інвестиції засновників	$4 000	на реєстрацію та дизайн

	Seed-раунд	$10 000	для розширення маркетингових активностей
Точка беззбитковості		9-12 місяць	при 25 000 безкоштовних юзерів і 5 % конверсії → $5 000/міс
Резерв непередбаченого		15 % бюджету	покриває додаткові витрати чи зростання вартості ресурсів

4.2.7 Оцінка ризиків та страхування

Успішна реалізація проєкту PixMatic неможлива без системного підходу до управління ризиками: їх раннього виявлення, оцінки впливу, розробки планів реагування та постійного моніторингу.

Згідно з PMBOK, цей процес охоплює п’ять ітеративних етапів: планування управління ризиками, ідентифікацію, аналіз, планування реакцій та моніторинг - і спрямований на мінімізацію негативних наслідків для термінів, вартості та якості продукту.

Основні ризики для реалізації проекту представлені у таблиці 4.3.

Таблиця 4.3

Основні ризики

Категорія	Опис
Технічні ризики	Уразливості через відмінності в реалізації Canvas API у різних браузерах, неочікувані помилки в алгоритмах, витоки пам’яті в JavaScript-рушії, що можуть призвести до зависань або краху редактора під час обробки великих зображень;
Ринкові ризики	Посилення конкуренції з боку безкоштовних онлайн-інструментів або нових функцій у Aseprite, зміни в запитах користувачів (наприклад, зростання попиту на інтеграцію з NFT-платформами) та коливання платоспроможності цільової аудиторії;

Операційні ризики	Проблеми у ключових спеціалістів (front-end, UX/UI), затримки у виплаті заробітних плат, недоліки в організації робочих процесів або недостатня комунікація між командою;
Фінансові та правові ризики	Непередбачені коливання цін на хостинг та CDN, затримки з грантовими виплатами, потенційні штрафи за недотримання GDPR та українського Закону «Про захист персональних даних»;

Процес управління ризиками:

1. Збір інформації через SWOT-аналіз, ідентифікація ризиків, перегляд документації та опитування членів команди і партнерів;
2. Кількісний та якісний аналіз, ранжування ризиків (матриця ризиків), оцінка потенційних грошових втрат та часу;
3. Планування реакцій та розробка заходів уникнення технічних проблем (наприклад додаткові автотести й ретестування в різних браузерах), зменшення ринкових ризиків;
4. Регулярний моніторинг реєстру ризиків, оновлення статусу, ініціювання навмисних епізодів (наприклад падіння FPS чи зріст часу відповіді API) для опрацювання планів реагування;

Висновки до розділу 4

Отже, аналізуючи четвертий розділ можна дійти висновку що принципи ергономіки інформаційних технологій і техніко-економічного обґрунтування створюють єдину і головне: надійну та логічну основу для розробки й впровадження веб-редактора піксель-арту PixMatic.

Поєднання дизайну, що ґрунтується на ISO 9241, з чітко вимірюваними метриками ефективності, продуктивності і задоволеності користувачів забезпечує виконавців управління проєкта розумінням потреб аудиторії та надає підказок у плануванні інтерфейсу, який згодом дійсно спростить робочі процеси користувачів-художників та розробників. Ітеративний цикл досліджень і тестування

дозволяє поступово виявляти й усувати вузькі місця, гарантує доступність і зрозумілість інструментів та знижує ймовірність помилок.

Детальний аналіз інвестицій, витрат і фінансових формує стійку економічну платформу для розвитку проєкту. На основі визначених ключових джерел фінансування та резервів на непередбачені витрати можливо побудувати реалістичний графік виходу на точку беззбитковості і забезпечити прозорість для інвесторів. Розроблена маркетингова стратегія з урахуванням фріміум-моделі та багатоканальних каналів просування створюють належні умови для успішного масштабування та довготривалої конкурентоспроможності PixMatic на ринку цифрових творчих інструментів.

Загалом використані підходи з ергономічного проєктування та фінансово-економічного планування не тільки окреслюють технічні й організаційні параметри розробки, а й формують цілісну стратегію, що гарантує ефективне впровадження, мінімізацію ризиків і найбільше задоволення потреб користувачів та партнерів. Насамперед, закладена надійна основа для подальшого зростання продукту, його інтеграції з популярними платформами та підтримки спільнотою, що зрештою забезпечить PixMatic тривалий успіх у сфері веб-редакторів піксель-арту.

ВИСНОВКИ

У процесі виконання дипломної роботи було повністю реалізовано задум створення веб-додатка PixMatic, призначеного для автоматизованого перетворення фотографій та ескізів у стиль піксель-арту. Застосувавши алгоритми адаптивного зменшення роздільної здатності та кластеризації кольорів з можливістю тонкого ручного редагування, проєкт успішно вирішив ключову проблематику ручної малювання в цьому стилі та істотно знизив часові витрати користувача. Логічно відокремлені модулі Image Loader, Canvas Controller, Processing Engine, UI Manager, State Manager і Export Module, побудовані на HTML5/CSS3 і JavaScript у межах спіральної моделі життєвого циклу, забезпечили високу чіткість та надійність коду, гнучке масштабування та зручність супроводу.

Практичні експерименти підтвердили ефективність реалізованих рішень: усі основні операції виконуються за час, що не перевищує 500 мс, причому складність зростає лінійно відносно кількості пікселів, а розбіжність результатів у різних браузерах не перевищує 10 %. Юзабіліті-тестування за участю незалежних респондентів продемонструвало інтуїтивність інтерфейсу та швидке освоєння інструментів, це в свою чергу демонструє досягнення ергономічних цілей. Фінансово-економічне обґрунтування окреслило реалістичний баланс CAPEX/OPEX, фріміум-стратегію монетизації та канали просування. Фінально маємо сформовану стійку платформу, що готова до подальшої комерціалізації й масштабування продукту.

Узагальнюючи результати, можна стверджувати, що поставлена мета досягнута повною мірою: створено конкурентоспроможний, технічно досконалий і економічно виправданий інструмент. PixMatic суттєво спрощує та прискорює процес генерації піксель-арту, розширює творчі можливості художників, навіть початківців, та підвищує репутаційний потенціал українських цифрових продуктів на світовому ринку. Отримані наукові й практичні результати відкривають перспективи для подальшого розвитку системи, а саме: інтеграції GPU-прискорення, розширення бібліотеки фільтрів, додавання адаптації під мобільні платформи - все це разом забезпечить довгострокову актуальність і конкурентоспроможність проєкту.

СПИСОК ВИКОРИСТАНИХ ЛІТЕРАТУРНИХ ДЖЕРЕЛ

1. Піксельна графіка [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%BA%D1%81%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0_%D0%B3%D1%80%D0%B0%D1%84%D1%96%D0%BA%D0%B0 (дата звернення 18.01.2025).
2. Пікселізація [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/%D0%9F%D1%96%D0%BA%D1%81%D0%B5%D0%BB%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%8F (дата звернення 18.01.2025).
3. Aseprite [Електронний ресурс] // Wikipedia. - URL: https://en.wikipedia.org/wiki/Aseprite (дата звернення 19.01.2025).
4. Piskel - free online sprite editor [Електронний ресурс] // GitHub. - URL: https://github.com/piskelapp/piskel (дата звернення 19.01.2025).
5. GrafX2 [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/GrafX2 (дата звернення 19.01.2025).
6. Adobe Photoshop [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/Adobe_Photoshop (дата звернення 19.01.2025).
7. Image size and resolution in Photoshop [Електронний ресурс] // Adobe HelpX. - URL: https://helpx.adobe.com/ua/photoshop/using/image-size-resolution.html#resampling (дата звернення 21.01.2025).
8. Масштабування зображення [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%88%D0%B0%D0%B1%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F_%D0%B7%D0

%BE%D0%B1%D1%80%D0%B0%D0%B6%D0%B5%D0%BD%D0%BD%D1%8F (дата звернення 22.01.2025).

9. Масштабування зображень [Електронний ресурс] // pzs.dstu.dp.ua. - URL: https://pzs.dstu.dp.ua/ComputerGraphics/scaling/index.html (дата звернення 30.01.2025).
10. Understanding Digital Image Interpolation [Електронний ресурс] // Cambridge in Colour. - URL: https://www.cambridgeincolour.com/tutorials/image-interpolation.htm (дата звернення 30.01.2025).
11. Methods of Image Interpolation [Електронний ресурс] // Tech & Science Journal (KSAU). - URL: https://journals.ksauniv.ks.ua/index.php/tech/article/view/446/414 (дата звернення 31.01.2025).
12. Білінійна інтерполяція [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/%D0%91%D1%96%D0%BB%D1%96%D0%BD%D1%96%D0%B9%D0%BD%D0%B0_%D1%96%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D0%BE%D0%BB%D1%8F%D1%86%D1%96%D1%8F (дата звернення 01.02.2025).
13. Python OpenCV - Bicubic Interpolation for Resizing Image [Електронний ресурс] // GeeksforGeeks. - URL: https://www.geeksforgeeks.org/python-opencv-bicubic-interpolation-for-resizing-image/ (дата звернення 02.02.2025).
14. K Means Clustering - Introduction [Електронний ресурс] // GeeksforGeeks. - URL: https://www.geeksforgeeks.org/k-means-clustering-introduction/ (дата звернення 10.02.2025).
15. k-means clustering algorithm [Електронний ресурс] // Google Sites. - URL: https://sites.google.com/site/dataclusteringalgorithms/k-means-clustering-algorithm (дата звернення 10.02.2025).

16. Median Cut Color Quantization [Електронний ресурс] // Joel Carlson Blog. - URL: https://joelcarlson.github.io/2016/01/15/median-cut (дата звернення 11.02.2025).
17. Octree [Електронний ресурс] // Wikipedia. - URL: https://en.wikipedia.org/wiki/Octree (дата звернення 11.02.2025).
18. Image Thresholding Tutorial [Електронний ресурс] // OpenCV Docs. - URL: https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html (дата звернення 12.02.2025).
19. HTML5 Canvas Tutorial [Електронний ресурс] // W3Schools. - URL: https://www.w3schools.com/html/html5_canvas.asp (дата звернення 25.02.2025).
20. CSS [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/CSS (дата звернення 25.02.2025).
21. VanillaJS vs Frameworks: When Using JS Libraries [Електронний ресурс] // Webix Blog. - URL: https://blog.webix.com/vanillajs-vs-framework-when-using-js-libraries (дата звернення 26.02.2025).
22. Canvas API Documentation [Електронний ресурс] // ED.Link Community. - URL: https://ed.link/community/canvas-api (дата звернення 27.02.2025).
23. Спіральна модель (Spiral Model) [Електронний ресурс] // QALight. - URL: https://qalight.ua/baza-znaniy/cpiralna-model-spiral-model (дата звернення 05.03.2025).
24. Architecture Styles Guide [Електронний ресурс] // Microsoft Learn. - URL: https://learn.microsoft.com/en-us/azure/architecture/guide/architecture-styles (дата звернення 06.03.2025).
25. Common Web Application Architectures [Електронний ресурс] // Microsoft Learn. - URL: https://learn.microsoft.com/en-us/dotnet/architecture/modern-web-apps-azure/common-web-application-architectures (дата звернення 06.03.2025).

26. Separation of Concerns (SoC) [Електронний ресурс] // GeeksforGeeks. - URL: https://www.geeksforgeeks.org/separation-of-concerns-soc (дата звернення 07.03.2025).

27. ES6 (JavaScript) Підручник [Електронний ресурс] // JavaScript.Tutorial.in.ua. - URL: https://javascript.tutorial.in.ua/es6 (дата звернення 08.03.2025).

28. Діаграма потоків даних [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA%D1%96%D0%B2_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85 (дата звернення 14.03.2025).

29. Responsive Web Design [Електронний ресурс] // Wikipedia. - URL: https://en.wikipedia.org/wiki/Responsive_web_design (дата звернення 14.03.2025).

30. Закон Фіттса: наука за лаштунками дизайну [Електронний ресурс] // DesignTalk.club. - URL: https://designtalk.club/nauka-za-lashtunkamy-dyzajnu-zakon-fittsa (дата звернення 20.03.2025).

31. Як протестувати сайт [Електронний ресурс] // Wezom Blog. - URL: https://wezom.com.ua/ua/blog/kak-protestirovat-sajt (дата звернення 21.03.2025).

32. Top 10 Test Design Techniques [Електронний ресурс] // Testing101. - URL: https://www.testing101.net/post/top-10-test-design-techniques (дата звернення 22.03.2025).

33. Human-Centered Design: How, When and Why [Електронний ресурс] // Computools Careers. - URL: https://careers.computools.ua/human-centered-design-how-when-and-why (дата звернення 23.03.2025).

34. Техніко-економічне обґрунтування [Електронний ресурс] // Вікіпедія: вільна енциклопедія. - URL: https://uk.wikipedia.org/wiki/%D0%A2%D0%B5%D1%85%D0%BD%D1%96%D0%BA%D0%BE-%D0%B5%D0%BA%D0%BE%D0%BD%D0%BE%D0%BC%D1%96%D1%87%D0%BD%D0%B5_%D0%BE%D0%B1%D2%91%D1%80%D1%83%D0%BD%D1%82%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F (дата звернення 10.04.2025).

35. Pixelorama - open-source pixel-art editor [Електронний ресурс] // Itch.io. - URL: https://orama-interactive.itch.io/pixelorama (дата звернення 20.04.2025).

36. Freemium Strategy: What It Is & How to Use It [Електронний ресурс] // Userpilot Blog. - URL: https://userpilot.com/blog/freemium-strategy (дата звернення 10.05.2025).

Додаток А. Теза учасника конференції

Автоматизований інструмент для перетворення фотографій або ескізів у піксель-арт

Анастасія Суткова, здобувач вищої освіти[1],
Олена Доля, доцент, доцент кафедри ІТ [1] (ORCID: 0000-0002-8320-4038)
[1] *Київський національний університет будівництва і архітектури, проспект Повітряних сил, 31, м.Київ, Україна*

АНОТАЦІЯ
У даній роботі розглядаються ключові аспекти розробки інструменту для перетворення фотографій або ескізів у піксель-арт. Проаналізовано, як впровадження такого цифрового рішення може покращити ефективність творчого процесу та розширити можливості художників і дизайнерів. Розглянуто ефективні підходи до створення алгоритмів автоматичного перетворення зображень у піксель-арт, а також інструментів для налаштування рівня деталізації та колірних палітр.

Ключові слова: перетворення зображення, піксель-арт, автоматизований інструмент

1. ВСТУП

У сучасному цифровому світі, автоматизація творчих процесів стала важливим інструментом для художників і дизайнерів. Одним із таких інструментів є система для перетворення фотографій або ескізів у піксель-арт. Цей стиль, популярний у відеоіграх і ретро-мистецтві, та в реалізації вимагає значної рутинної ручної праці, що у свою чергу займає нераціонально багато часу. Тож, створення автоматизованого інструменту для конвертації зображень дозволить прискорити процес створення піксельного малюнку, зберігаючи його якість та загалом художню цінність.

2. АВТОМАТИЗАЦІЯ ПРОЦЕСУ СТВОРЕННЯ ПІКСЕЛЬ-АРТУ

Автоматизація процесу створення піксель-арту значно спрощує перетворення складних зображень у стилізовану ретро-графіку, що зменшує трудомісткість та час на виконання завдань. Для цього використовуються алгоритми, які аналізують фотографії або ескізи, спрощують їх структуру та адаптують під обмежену палітру

кольорів і низьку роздільну здатність, характерну для піксель-арту. Такий підхід дозволяє не лише художникам та розробникам ігор швидко створювати контент, а й сприяє стандартизації процесу, підвищуючи загальну якість графіки [1]. Автоматичні інструменти можуть пропонувати налаштування рівня деталізації, що робить їх універсальними як для професіоналів, так і для початківців, дозволяючи їм легко створювати піксельні зображення без глибоких знань або досвіду в ручній графіці [2].

3. ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ-ЗАСТОСУНКУ

Програмна реалізація інструменту для перетворення зображень у піксель-арт вимагає ретельного вибору технологій та методів. Важливо забезпечити ефективність алгоритмів, зручність інтерфейсу та надійність роботи на різних платформах.

3.1. Вибір технологічного стека

Для обчислювального ядра веб-застосунку доцільно використовувати нативну зв'язку HTML5 + Canvas API у поєднанні з Vanilla JavaScript. Canvas API [3] забезпечує прямий доступ до піксельних даних, що дає змогу втілити алгоритми зменшення роздільної здатності, кластеризацію кольорів та фільтрацію без сторонніх бібліотек. Використання OffscreenCanvas[4] і Web Workers зменшує блокування головного потоку, збільшуючи продуктивність під час обробки великих зображень у реальному часі. Допоміжно можна підключити WASM-модулі (наприклад, компільований із C/C++) для ресурсомістких операцій, лишаючи логіку редактора повністю у браузерному середовищі, що полегшує розгортання та масштабування.

Для зберігання даних доцільно застосувати IndexedDB [5] - документно-орієнтоване клієнтське сховище, яке дозволяє кешувати проєкти, палітри та налаштування у форматі JSON-подібних об'єктів без залучення окремого серверу. За потреби синхронізації між пристроями можна інтегрувати File System Access API чи хмарні бекенди (Firebase Storage, Dropbox Chooser), зберігаючи при цьому офлайн-спроможність редактора.

На клієнтському рівні інтерфейс реалізовано за допомогою Flexbox і CSS Grid із застосуванням кастомних Web Components. Ці технології формують компонентну архітектуру, що спрощує повторне використання елементів UI - панелей інструментів, слайдерів та діалогів експорту. Shadow DOM ізолює стилі кожного компонента, запобігаючи конфліктам, а використання ARIA-атрибутів забезпечує доступність інтерфейсу. Завдяки Virtual DOM Canvas-орієнтовані операції виконуються без перемальовування непотрібних вузлів, забезпечуючи плавну роботу навіть під час масштабного редагування. Для керування параметрами користувач отримує інтерактивні слайдери, випадаючі меню та гарячі клавіші, що дозволяє в реальному часі задавати розмір пікселів, глибину палітри та вибір додаткових фільтрів.

3.2. Розробка компонентів серверної частини

Серверна частина відповідає за обробку зображень, збереження проектів користувачів та інтеграцію з хмарними сховищами.Основні компоненти включають:

- **Завантаження та обробка зображень.** Сервер повинен мати функціонал для обробки зображень, який включає зменшення роздільної здатності для пікселізації та застосування фільтрів для налаштування кольорової гами. Цей процес виконується за допомогою бібліотеки Pillow або OpenCV, що дозволяє проводити обробку зображень на сервері.
- **Зберігання файлів та проектів.** Після обробки зображення та його конвертації в піксель-арт, користувач може зберігати свої роботи на сервері. Для цього використовуються бази даних, такі як MongoDB, що дозволяє зберігати як вихідні зображення, так і налаштування для подальшого редагування. Крім того, можна використовувати хмарні сховища, такі як Amazon S3, для зберігання великих файлів.

- **Інтегрованість з іншими сервісами.** Наприклад, інтеграція з хмарними сервісами для збереження даних і з платформами для спільного редагування дозволить розширити можливості інструменту.

3.3. Розробка компонентів клієнтської частини

Клієнтська частина — це інтерфейс, з яким взаємодіє користувач для завантаження зображень, налаштування параметрів пікселізації та перегляду результатів. Основні функції інтерфейсу:

- **Попередній перегляд результату.** Важливо забезпечити функцію попереднього перегляду, щоб користувач міг побачити, як виглядатиме кінцеве зображення після застосування фільтрів та налаштувань.
- **Можливість збереження та експорту.** Після завершення роботи користувач може зберегти результати на свій комп'ютер або поділитися ними через соціальні мережі чи інші платформи.
- **Завантаження зображень та налаштування параметрів.** Користувач повинен мати можливість завантажити зображення у форматах JPEG, PNG, GIF. Інтерфейс має включати інструменти для вибору розміру пікселів, кількості кольорів у палітрі та інших параметрів, таких як різкість або контрастність.

3.4. Інтеграція сторонніх сервісів

Для покращення функціональності інструменту важливо забезпечити інтеграцію зі сторонніми сервісами, цього можна досягти застосовуючи хмарні сервіси для зберігання та синхронізації проектів, наприклад, Google Drive або Dropbox, щоб користувачі могли зберігати свої файли безпосередньо у хмарі та мати доступ до них з будь-якого пристрою. Також для подальшого редагування результатів доцільно буде використовувати API графічних редакторів, таких як Adobe Photoshop або GIMP. Інтеграція з соціальними мережами дозволить користувачам ділитися своїми роботами безпосередньо з інтерфейсу програми.

3.5. Хостинг

Для стабільної роботи інструменту важливо вибрати відповідне середовище для хостингу. Використання хмарних платформ, таких як Amazon Web Services (AWS) [4], дозволить автоматично масштабувати ресурси в залежності від навантаження на систему. Це особливо важливо, оскільки піксель-арт може бути затребуваним серед великої аудиторії.

Хмарні рішення забезпечують також управління базами даних і зберігання файлів користувачів, що дозволяє налаштувати автоматичне резервне копіювання даних і швидке відновлення у випадку збоїв. Використання CDN-сервісів (Cloudflare або Fastly) дозволить прискорити завантаження веб-додатку для користувачів з різних регіонів.

3.6. Безпека та резервне копіювання

З огляду на те, що інструмент працює з персональними зображеннями користувачів, важливо забезпечити високий рівень безпеки. Це включає шифрування даних за допомогою SSL-сертифікатів, впровадження безпечної аутентифікації через OAuth або JWT, а також захист від атак, таких як SQL-ін'єкції та XSS.

Крім того, регулярне резервне копіювання даних є ключовим для збереження файлів користувачів. Використання хмарних рішень для автоматичного резервування дозволяє уникнути загрози втрати даних.

3.7. Тестування та підтримка

Перед випуском інструменту необхідно провести ретельне тестування його роботи. Це включає перевірку функціонування алгоритмів обробки зображень, зручність інтерфейсу користувача та продуктивність системи під високим навантаженням. Після запуску важливо забезпечити регулярне оновлення та додавання нових функцій на основі відгуків користувачів. та нових технологічних вимог.

4. ВИСНОВКИ

Розробка автоматизованого інструменту для перетворення фотографій або ескізів у піксель-арт є важливим кроком у спрощенні творчого процесу. Вибір правильного

технологічного стека, розробка ефективних алгоритмів та забезпечення зручного інтерфейсу дозволять створити інструмент, який буде корисним для художників та дизайнерів. Такий підхід допоможе зробити процес створення піксельної графіки швидким і доступним, зберігаючи при цьому високу якість результату та сприяючи створенню нових видів цифрового мистецтва.

Список літератури

[1] CADVision “Latest trends in Design Process Automation”. https://cadvisionengineers.com/latest-trends-in-design-process-automation/

[2] Pixelixe “Boosting Creativity - How Graphic Automation Frees Up Time for Designers”. https://pixelixe.com/blog/how-graphic-automation-frees-up-time-for-designers/

[3] MDN Web Docs. «Pixel manipulation with canvas». https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas

[4] web.dev**.** «OffscreenCanvas — speed up your canvas operations with a web worker».

https://web.dev/articles/offscreen-canvas

[5] MDN Web Docs. «IndexedDB API». https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API

[6] Spot “AWS Auto Scaling: Scaling EC2, ECS, RDS, and more”. https://spot.io/resources/aws-autoscaling/scaling-ec2-ecs-rds-and-more/

Додаток Б. Код розробки

HTML структура:

```
<!DOCTYPE html>
<html lang="uk">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>PixMatic</title>
    <link rel="stylesheet" href="styles.css">
    <link rel="icon" href="img/logo_icon.ico" type="image/x-icon">
</head>
<body>
    <div class="toolbar">
        <div class="logo"></div>
        <label for="upload" class="btn-upload"></label>
        <input type="file" id="upload" accept="image/*" onchange="loadImage(event)"
style="display: none;">
        <label>Interpolation</label>
        <input type="range" id="resizeRange" min="1" max="100" value="100"
onchange="resizeImage(this.value)">
        <div id="resolutionDisplay">[ 0×0 pixel ]</div>
        <label>Clasterization </label>
        <input type="range" id="clusterRange" min="2" max="100" value="100"
onchange="clusterColors(this.value)">
        <button onclick="toggleFilterMenu()" class="btn-filter"></button>
        <div id="filterMenu" class="filter-menu" style="display: none;">
        <button onclick="selectFilter('gaussian')">Gaussian blur</button>
        <button onclick="selectFilter('posterize')">Posterization</button>
        <button onclick="selectFilter('sobel')">Sobel contouring</button>
        <button onclick="selectFilter('cartoon')">Cartoon-effect</button>
        <button onclick="selectFilter('mono')">Monochrome</button>
        <button onclick="selectFilter('bright')">Bright</button>
        <button onclick="selectFilter('pixel')">Pixel effect</button>
        <button onclick="selectFilter('retro')">Retro style</button>
        <button onclick="selectFilter('glitch')">Glitch</button>
        <button onclick="selectFilter('dither')">Dithering</button>
        <button onclick="selectFilter('2bit')">2-bit</button>
        <br/>
```

```
        <div class="filter-actions">
        <button onclick="closeFilterMenu()">✕</button>
        <button onclick="applySelectedFilter()">Apply</button>
        </div>
        </div>
        <div class="panel-bottom" id="panelBottom">
            <div class="panel-text" id="panelText">Please, firstly download your
picture.</div>
        </div>
        <div class="panel-top" id="panelTop"></div>
        <button class= "ok-button" id="okButton"></button>
        <div class="tools_holder">
        <div class="tools1">
            <button onclick="setTool('brush')" class="btn-brush"></button>
            <button onclick="setTool('eraser')" class="btn-eraser"></button>
            <button onclick="setTool('fill')" class="btn-fill"></button>
        </div>
        <div class="tools2">
            <button onclick="setTool('blur')"  class="btn-blur"></button>
            <button onclick="setTool('spray')" class="btn-spray"></button>
            <button onclick="setTool('dither')" class="btn-dither"></button>
        </div>
        <div class="tools3">
        <button onclick="setTool('line')"    class="btn-line"></button>
        <button onclick="setTool('rect')"    class="btn-rect"></button>
        <button onclick="setTool('ellipse')" class="btn-ellipse"></button>
        </div>

        </div>
        <div class="tool-settings">
          <label for="brushSizeSlider">Size:</label>
          <input type="range" id="brushSizeSlider" min="1" max="10" value="1">
          <span id="brushSizeLabel">1x</span>
        </div>
        <div id="activeColorBar" class="active-color-bar">
            <div id="activeColor" class="active-color"></div>
        </div>
        <input type="color" id="colorInput" style="display: none;"/>
        <div class="palette-container">
```

```
            <div id="colorPalette" class="color-palette"></div>
        </div>
        <div class="arrows">
            <button onclick="undo()" class="btn-undo"></button>
            <div class="action">
                <button id="showBaseBtn" class="btn-action-show"></button>
                <button id="resetBtn" class="btn-action-canceling"></button>
            </div>
            <button onclick="redo()" class="btn-redo"></button>
        </div>

<button onclick="openExportModal()" class="btn-download"></button>
        <div id="exportModal" class="modal" style="display: none;">
            <div class="modal-content">
              <span class="close" onclick="closeExportModal()">×</span>
              <h1>Size settings</h1>
              <label>Width (px):</label>
              <input type="number" id="exportWidth"
oninput="syncExportHeight('width')"/><br/>
              <label>Height (px):</label>
              <input type="number" id="exportHeight"
oninput="syncExportHeight('height')"/><br/>
              <label>Format:</label>
              <select id="exportFormat">
                <option value="png" selected>PNG</option>
                <option value="jpg">JPG</option>
                <option value="gif">GIF</option>
                <option value="svg">SVG</option>
                <option value="ai">AI</option>
              </select>
              <br/>
              <button onclick="confirmExport()"
class="modal-ok-button"><h2>OK</h2></button>
            </div></div></div>

    <div class="canvas-container">
        <div class="canvas-wrapper" id="canvasWrapper">
          <canvas id="canvas" width="500" height="500"></canvas>
         </div>
```

```
    </div>
    <div id="cursorPos" class="cursor-pos">X: 0; Y: 0;</div>

    <script src="script.js"></script>
</body>
</html>
```

JavaScript завантаження оригінального зображення:

```
/// Завантаження зображення та процеси
function loadImage(event) {
    const file = event.target.files[0];
    if (!file) return;
    const reader = new FileReader();
    reader.onload = (e) => {
        const img = new Image();
        img.onload = () => {
            canvas.width = img.width;
            canvas.height = img.height;
            ctx.clearRect(0, 0, canvas.width, canvas.height);
            ctx.drawImage(img, 0, 0);
            baseImage = ctx.getImageData(0, 0, canvas.width, canvas.height);
            originalImage = img;
            processedImage = img;
            viewScale = 1;
            imageLoaded = true;
            panelText.textContent = "Change the ranges and tap 'ok'."
            centerCanvas();
        };
        img.src = e.target.result;
    };
    reader.readAsDataURL(file);
}
```

JavaScript масштабування та кластеризація кольорів зображення:

```
/// Інтерполяція, Масштаб, K-Means clustering з відбором медоїдів

function resizeImage(scale) {
```

```
    if (!imageLoaded) return;
    if (!originalImage) return;
    const truncWidth = Math.floor(originalImage.width / 10) * 10;
    const truncHeight = Math.floor(originalImage.height / 10) * 10;
    const scaleFactor = scale / 25;
    const logicalWidth = Math.round(truncWidth * scaleFactor);
    const logicalHeight = Math.round(truncHeight * scaleFactor);
    if (logicalWidth < 1 || logicalHeight < 1) return;
    const truncCanvas = document.createElement("canvas");
    truncCanvas.width = truncWidth;
    truncCanvas.height = truncHeight;
    const truncCtx = truncCanvas.getContext("2d");
    truncCtx.imageSmoothingEnabled = false;
    truncCtx.drawImage(originalImage, 0, 0, truncWidth, truncHeight);
    const logicalCanvas = document.createElement("canvas");
    logicalCanvas.width = logicalWidth;
    logicalCanvas.height = logicalHeight;
    const logicalCtx = logicalCanvas.getContext("2d");
    logicalCtx.imageSmoothingEnabled = false;
    logicalCtx.drawImage(
        truncCanvas,
        0, 0, truncWidth, truncHeight,
        0, 0, logicalWidth, logicalHeight
    );

    canvas.width = logicalWidth;
    canvas.height = logicalHeight;
    ctx.imageSmoothingEnabled = false;
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.drawImage(logicalCanvas, 0, 0);

    processedImage = new Image();
    processedImage.src = logicalCanvas.toDataURL();
    pixelSize = 1;
    autoScale();
    updateResolutionDisplay(logicalWidth, logicalHeight);
}

function clusterColors(level) {
```

```
if (!originalImage || !imageLoaded) return;
const width  = canvas.width;
const height = canvas.height;

if (width > 1000 || height > 1000) {
    alert("Зображення завелике, кластеризація не відбуватиметься!");
    return;
}
const tempCanvas = document.createElement('canvas');
tempCanvas.width = canvas.width;
tempCanvas.height = canvas.height;
const tempCtx = tempCanvas.getContext('2d');
tempCtx.imageSmoothingEnabled = false;
tempCtx.drawImage(originalImage, 0, 0, canvas.width, canvas.height);

const imageData = tempCtx.getImageData(0, 0, canvas.width, canvas.height);
const data = imageData.data;
const pixels = [];
for (let i = 0; i < data.length; i += 4) {
    pixels.push([data[i], data[i+1], data[i+2]]);
}

const k = Math.min(level, pixels.length);
const maxIter = 10;
let centroids = [];
const used = new Set();
while (centroids.length < k) {
    let idx = Math.floor(Math.random() * pixels.length);
    if (!used.has(idx)) {
        used.add(idx);
        centroids.push(pixels[idx].slice());}}
const assignments = new Array(pixels.length);

for (let iter = 0; iter < maxIter; iter++) {
    let moved = false;
    for (let i = 0; i < pixels.length; i++) {
        let minDist = Infinity, cluster = 0;
        for (let c = 0; c < k; c++) {
            const dx = pixels[i][0] - centroids[c][0];
```

```
            const dy = pixels[i][1] - centroids[c][1];
            const dz = pixels[i][2] - centroids[c][2];
            const d2 = dx*dx + dy*dy + dz*dz;
            if (d2 < minDist) {
                minDist = d2;
                cluster = c; }}
        if (assignments[i] !== cluster) {
            moved = true;
            assignments[i] = cluster;
        }}
    if (!moved) break;

    const sums = Array.from({length: k}, () => [0,0,0]);
    const counts = Array(k).fill(0);
    for (let i = 0; i < pixels.length; i++) {
        const c = assignments[i];
        sums[c][0] += pixels[i][0];
        sums[c][1] += pixels[i][1];
        sums[c][2] += pixels[i][2];
        counts[c]++;
    }
    for (let c = 0; c < k; c++) {
        if (counts[c] > 0) {
            centroids[c][0] = sums[c][0] / counts[c];
            centroids[c][1] = sums[c][1] / counts[c];
            centroids[c][2] = sums[c][2] / counts[c];
        }}}
const palette = centroids.map((centroid, c) => {
    let minDist = Infinity;
    let medoid = centroid;
    for (let i = 0; i < pixels.length; i++) {
        if (assignments[i] !== c) continue;
        const dx = pixels[i][0] - centroid[0];
        const dy = pixels[i][1] - centroid[1];
        const dz = pixels[i][2] - centroid[2];
        const d2 = dx*dx + dy*dy + dz*dz;
        if (d2 < minDist) {
            minDist = d2;
            medoid = pixels[i]; }} return medoid; });
```

```
    for (let i = 0, j = 0; i < pixels.length; i++, j += 4) {
        const [r,g,b] = palette[assignments[i]];
        data[j]   = r;
        data[j+1] = g;
        data[j+2] = b;
    }
    tempCtx.putImageData(imageData, 0, 0);
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.drawImage(tempCanvas, 0, 0);
    processedImage = new Image();
    processedImage.src = tempCanvas.toDataURL();
}
```

JavaScript фільтри:

```
// Меню фільтрів та самі фільтри

let filterMenuVisible = false;
let chosenFilter = null;
let backupImage = null;
function toggleFilterMenu() {
    if (!imageLoaded) return;
    const menu = document.getElementById('filterMenu');
    if (filterMenuVisible) {
      revertToBackupAndCloseMenu();
      return;
    }
    filterMenuVisible = true;
    menu.style.display = 'block';
    chosenFilter = null;
    backupImage = new Image();
    backupImage.src = processedImage.src;
  }
function selectFilter(name) {
    chosenFilter = name;
    revertToBackupImageOnly();
    applyFilter(chosenFilter);
}
```

```
function closeFilterMenu() {
    revertToBackupAndCloseMenu();
}
function applySelectedFilter() {
    if (!chosenFilter) {
      revertToBackupAndCloseMenu();
      return;
    }
    document.getElementById('filterMenu').style.display = 'none';
    filterMenuVisible = false;
    chosenFilter = null;
    backupImage = null;
}
function revertToBackupAndCloseMenu() {
    if (backupImage) {
      revertToBackupImageOnly();
    }
    document.getElementById('filterMenu').style.display = 'none';
    filterMenuVisible = false;
    chosenFilter = null;
    backupImage = null;
}
function revertToBackupImageOnly() {
    if (!backupImage) return;
    processedImage = new Image();
    processedImage.src = backupImage.src;
    processedImage.onload = () => {
      ctx.clearRect(0, 0, canvas.width, canvas.height);
      ctx.drawImage(processedImage, 0, 0, canvas.width, canvas.height);
    };
}
function applyFilter(filterType) {
    if (!processedImage) return;
    const tempCanvas = document.createElement('canvas');
    tempCanvas.width = canvas.width;
    tempCanvas.height = canvas.height;
    const tempCtx = tempCanvas.getContext('2d');
    tempCtx.imageSmoothingEnabled = false;
    tempCtx.drawImage(processedImage, 0, 0, canvas.width, canvas.height);
```

```
let imageData = tempCtx.getImageData(0, 0, tempCanvas.width, tempCanvas.height);

switch(filterType) {
  case 'gaussian':
    imageData = gaussianBlur(imageData);
    break;
  case 'posterize':
    imageData = posterize(imageData);
    break;
  case 'sobel':
    imageData = sobelEdge(imageData);
    break;
  case 'cartoon':
    imageData = cartoonEffect(imageData);
    break;
  case 'mono':
    imageData = monoEffect(imageData);
    break;
  case 'bright':
    imageData = brightEffect(imageData);
    break;
  case 'pixel':
    imageData = pixelEffect(imageData);
    break;
    case 'retro':
    imageData = retroPaletteEffect(imageData);
    break;
    case 'glitch':
    imageData = glitchEffect(imageData);
    break;
    case 'dither':
    imageData = floydSteinbergDither(imageData);
    break;
    case '2bit':
    imageData = twoBitEffect(imageData);
    break;
  default:
    console.warn('X', filterType);
    return; }
```

```
    tempCtx.putImageData(imageData, 0, 0);
    ctx.clearRect(0, 0, canvas.width, canvas.height);
    ctx.drawImage(tempCanvas, 0, 0);
    processedImage = new Image();
    processedImage.src = tempCanvas.toDataURL();}


function monoEffect(imageData) {
    const d = imageData.data;
    for (let i=0; i<d.length; i+=4){
      const avg = (d[i] + d[i+1] + d[i+2]) /3;
      d[i] = d[i+1] = d[i+2] = avg;
    }
    return imageData;
}
function brightEffect(imageData) {
    const d = imageData.data;
    for (let i=0; i<d.length; i+=4){
      d[i]   = Math.min(d[i] + 50, 255);
      d[i+1] = Math.min(d[i+1] + 50, 255);
      d[i+2] = Math.min(d[i+2] + 50, 255);
    }return imageData;
}
function posterize(imageData) {
    const d = imageData.data;
    for (let i=0; i<d.length; i+=4){
      d[i]   = Math.floor(d[i]   / 32)*32;
      d[i+1] = Math.floor(d[i+1] / 32)*32;
      d[i+2] = Math.floor(d[i+2] / 32)*32;
    } return imageData;
}

function cartoonEffect(imageData) {
    imageData = convolution(imageData, [1,1,1,1,1,1,1,1,1], 9);
    for (let i=0; i<imageData.data.length; i+=4){
      imageData.data[i]   = Math.floor(imageData.data[i]/64)*64;
      imageData.data[i+1] = Math.floor(imageData.data[i+1]/64)*64;
      imageData.data[i+2] = Math.floor(imageData.data[i+2]/64)*64;
    }return imageData;
```

```
}
function sobelEdge(imageData) {
    const gray = monoEffect(imageData);
    const gx = [-1,0,1, -2,0,2, -1,0,1];
    const gy = [-1,-2,-1, 0,0,0, 1,2,1];
    const dataX = convolution(gray, gx, 1);
    const dataY = convolution(gray, gy, 1);
    const out = dataX;
    for (let i=0; i<out.data.length; i+=4){
      const px = dataX.data[i];
      const py = dataY.data[i];
      const v = Math.sqrt(px*px + py*py);
      out.data[i] = out.data[i+1] = out.data[i+2] = Math.min(v, 255);
    } return out;
}
function pixelEffect(imageData) {
    for (let i=0; i<imageData.data.length; i+=4){
      imageData.data[i]   = Math.floor(imageData.data[i]/32)*32;
      imageData.data[i+1] = Math.floor(imageData.data[i+1]/32)*32;
      imageData.data[i+2] = Math.floor(imageData.data[i+2]/32)*32;
    }return imageData;
}
function retroPaletteEffect(imageData) {
    const d = imageData.data;
    for (let i=0; i<d.length; i+=4) {
      d[i]   = Math.floor(d[i]   /16)*16;
      d[i+1] = Math.floor(d[i+1] /16)*16;
      d[i+2] = Math.floor(d[i+2] /16)*16;
    }return imageData;
}
function glitchEffect(imageData) {
    const w = imageData.width;
    const h = imageData.height;
    const d = imageData.data;
    for (let y = 0; y < h; y += 10) {
      const shift = Math.floor(Math.random()*11) - 5;
      if (shift === 0) continue;
      for (let x = 0; x < w - shift; x++) {
        const srcPos = ((y*w) + x)*4;
```

```
        const dstPos = ((y*w) + (x+shift))*4;
        d[dstPos]   = d[srcPos];
        d[dstPos+1] = d[srcPos+1];
        d[dstPos+2] = d[srcPos+2];
        d[dstPos+3] = d[srcPos+3];
      }}return imageData;
}
function floydSteinbergDither(imageData) {
    const w = imageData.width;
    const h = imageData.height;
    const d = imageData.data;

    for (let y=0; y<h; y++){
      for (let x=0; x<w; x++){
        const i = (y*w + x)*4;
        const oldR = d[i];
        const oldG = d[i+1];
        const oldB = d[i+2];
        const newR = Math.round(oldR/85)*85;
        const newG = Math.round(oldG/85)*85;
        const newB = Math.round(oldB/85)*85;
        d[i]   = newR;
        d[i+1] = newG;
        d[i+2] = newB;
        const errR = oldR - newR;
        const errG = oldG - newG;
        const errB = oldB - newB;
        spreadError(x+1, y,   7/16, errR, errG, errB, d, w, h);
        spreadError(x-1, y+1, 3/16, errR, errG, errB, d, w, h);
        spreadError(x,   y+1, 5/16, errR, errG, errB, d, w, h);
        spreadError(x+1, y+1, 1/16, errR, errG, errB, d, w, h);
      }} return imageData;
}
function spreadError(x, y, factor, errR, errG, errB, data, w, h) {
    if (x<0 || x>=w || y<0 || y>=h) return;
    const i = (y*w + x)*4;
    data[i]   = clamp(data[i]   + errR*factor);
    data[i+1] = clamp(data[i+1] + errG*factor);
    data[i+2] = clamp(data[i+2] + errB*factor);
```

```
}
function clamp(v) {
    return Math.max(0, Math.min(255, v));
}
function twoBitEffect(imageData) {
    const d = imageData.data;
    for (let i=0; i<d.length; i+=4){
      const gray = 0.299*d[i] + 0.587*d[i+1] + 0.114*d[i+2];
      let level = 0;
      if (gray > 191) level = 255;
      else if (gray > 127) level = 170;
      else if (gray > 63)  level = 85;
      else level = 0;
      d[i] = d[i+1] = d[i+2] = level;
    }return imageData;
}
function gaussianBlur(imageData) {
  return convolution(imageData, gaussian5x5Kernel, 273);
}
const gaussian5x5Kernel = [
    1,  4,  7,  4,  1,
    4, 16, 26, 16,  4,
    7, 26, 41, 26,  7,
    4, 16, 26, 16,  4,
    1,  4,  7,  4,  1
];
function convolution(imageData, kernel, divisor) {
    const w = imageData.width;
    const h = imageData.height;
    const src = imageData.data;
    const output = new Uint8ClampedArray(src.length);
    const side = Math.sqrt(kernel.length);
    const half = Math.floor(side/2);
    for (let y=0; y<h; y++){
      for (let x=0; x<w; x++){
        let r=0, g=0, b=0;
        for (let ky=0; ky<side; ky++){
          for (let kx=0; kx<side; kx++){
            const px = x + (kx - half);
```

```
            const py = y + (ky - half);
            if (px>=0 && px<w && py>=0 && py<h){
              const offset = (py*w + px)*4;
              const wt = kernel[ky*side + kx];
              r += src[offset]   * wt;
              g += src[offset+1] * wt;
              b += src[offset+2] * wt;
            }}}
          const i = (y*w + x)*4;
          output[i]   = Math.min(Math.max(r/divisor,0),255);
          output[i+1] = Math.min(Math.max(g/divisor,0),255);
          output[i+2] = Math.min(Math.max(b/divisor,0),255);
          output[i+3] = 255;
        }
      }
      const out = new ImageData(w, h);
      out.data.set(output);
      return out;}
```

JavaScript історія подій:

```
// Історія подій, Вперед/Назад
let history = [];
let historyIndex = -1;
const maxHistory = 100;
let lastX = null, lastY = null;
let currentStroke = null;
function startStroke(){
    currentStroke=[];
}
function endStroke(){
    if(!currentStroke || currentStroke.length===0){
      currentStroke=null;
      return;
    }
    const ev={
      type: tool,
      size: pixelSize,
      changes:currentStroke
```

```
    };
    pushHistoryEvent(ev);
    currentStroke=null;
    lastX=null;
    lastY=null;
}
function pushHistoryEvent(event){
    if(historyIndex< history.length-1){
      history.splice(historyIndex+1);
    }
    history.push(event);
    if(history.length>maxHistory) history.shift();
    historyIndex=history.length-1;
}
function undo(){
    if(historyIndex<0) return;
    const ev=history[historyIndex];
    revertChanges(ev);
    historyIndex--;
}
function redo(){
    if(historyIndex>=history.length-1) return;
    historyIndex++;
    const ev=history[historyIndex];
    applyChanges(ev);
}
function revertChanges(ev) {
    ev.changes.forEach(c => {
            if (c.oldBlock) {
              ctx.putImageData(c.oldBlock, c.x, c.y);
    } else {
            const [r,g,b,a] = c.oldColor;
            ctx.fillStyle = `rgba(${r},${g},${b},${a/255})`;
            ctx.fillRect(c.x, c.y, 1, 1);
        }
    });}
function applyChanges(ev){
    ev.changes.forEach(c => {
            if (c.newBlock) {
```

```
            ctx.putImageData(c.newBlock, c.x, c.y);
        } else {
            const [r,g,b,a] = c.newColor;
            ctx.fillStyle = `rgba(${r},${g},${b},${a/255})`;
            ctx.fillRect(c.x, c.y, 1, 1);
        }    });}
function applyPixelWithHistory(px, py) {
        const isErase = (tool === 'eraser');
        const newColor = isErase ? [0,0,0,0] : parseColor(color);

        for (let yy = 0; yy < pixelSize; yy++) {
          for (let xx = 0; xx < pixelSize; xx++) {
            const x = px + xx, y = py + yy;
            const oldColor = getPixelColor(x, y);
            if (colorsMatch(oldColor, newColor)) continue;
            if (isErase) {
              ctx.clearRect(x, y, 1, 1);
            } else {
              ctx.fillStyle =
`rgba(${newColor[0]},${newColor[1]},${newColor[2]},${newColor[3]/255})`;
              ctx.fillRect(x, y, 1, 1);
            }
            if (currentStroke) {
              currentStroke.push({ x, y, oldColor, newColor });
            }}}}
function getPixelColor(x,y){
    if(x<0||y<0||x>=canvas.width||y>=canvas.height) return [0,0,0,0];
    const imgd=ctx.getImageData(x,y,1,1).data;
    return [imgd[0], imgd[1], imgd[2], imgd[3]];
}
function applyPixel(x,y, rgba){
    for(let yy=0; yy<pixelSize; yy++){
      for(let xx=0; xx<pixelSize; xx++){
        const rx = x+xx, ry = y+yy;
        if(rx<0||ry<0||rx>=canvas.width||ry>=canvas.height) continue;
        let imgd = ctx.getImageData(rx, ry, 1,1);
        let d=imgd.data;
        d[0]=rgba[0];
        d[1]=rgba[1];
```

```
        d[2]=rgba[2];
        d[3]=rgba[3];
        ctx.putImageData(imgd, rx, ry);
      }}}
```

JavaScript збереження редагованого зображення:

```
// Збереження зображення
let exportOriginalW = 0;
let exportOriginalH = 0;
let exportRatio = 1.0;
function openExportModal() {
  const modal = document.getElementById("exportModal");
  modal.style.display = "flex";
  exportOriginalW = canvas.width;
  exportOriginalH = canvas.height;
  exportRatio = exportOriginalW / exportOriginalH;

  document.getElementById("exportWidth").value = exportOriginalW;
  document.getElementById("exportHeight").value = exportOriginalH;
}
function closeExportModal() {
  const modal = document.getElementById("exportModal");
  modal.style.display = "none";
}
function syncExportHeight(changedField) {
  const wField = document.getElementById("exportWidth");
  const hField = document.getElementById("exportHeight");
  let wVal = parseInt(wField.value, 10);
  let hVal = parseInt(hField.value, 10);
  if (wVal < 1) wVal = 1;
  if (hVal < 1) hVal = 1;
  if (changedField === 'width') {
    const newHeight = Math.round(wVal / exportRatio);
    hField.value = newHeight;
  } else {
    const newWidth = Math.round(hVal * exportRatio);
    wField.value = newWidth;
```

```
  }
}


function confirmExport() {
    const w = parseInt(document.getElementById("exportWidth").value,10)||1;
    const h = parseInt(document.getElementById("exportHeight").value,10)||1;
    const format = document.getElementById("exportFormat").value;
    const tempCanvas = document.createElement("canvas");
    tempCanvas.width = w;
    tempCanvas.height = h;
    const tctx = tempCanvas.getContext("2d");
    tctx.imageSmoothingEnabled = false;
    tctx.drawImage(canvas, 0,0, canvas.width, canvas.height, 0,0, w,h);

    switch(format){
        case "png": saveAsPng(tempCanvas);  break;
        case "jpg": saveAsJpg(tempCanvas);  break;
        case "gif": saveAsGif(tempCanvas);  break;
        case "svg": saveAsSvg(tempCanvas);  break;
        case "ai":  saveAsAi(tempCanvas);   break;
        default:
          alert("Формат не підтримується, зберігаємо PNG");
          saveAsPng(tempCanvas);
      }

      closeExportModal();
}
  function saveAsPng(canvasEl) {
    const dataURL = canvasEl.toDataURL("image/png");
    downloadDataURL(dataURL, "pixmatic.png");
  }
  function saveAsJpg(canvasEl) {
    const dataURL = canvasEl.toDataURL("image/jpeg");
    downloadDataURL(dataURL, "pixmatic.jpg");
  }
  function saveAsGif(canvasEl) {
    const dataURL = canvasEl.toDataURL("image/png");
    downloadDataURL(dataURL, "pixmatic.gif");
```

```
  }
  function saveAsSvg(canvasEl) {
    const dataURL = canvasEl.toDataURL("image/png");
    const w = canvasEl.width;
    const h = canvasEl.height;
    const svgContent =
  `<svg version="1.1" xmlns="http://www.w3.org/2000/svg" width="${w}" height="${h}">
    <image x="0" y="0" width="${w}" height="${h}" xlink:href="${dataURL}"
xmlns:xlink="http://www.w3.org/1999/xlink"/>
  </svg>`;
    const blob = new Blob([svgContent], { type: "image/svg+xml" });
    const url = URL.createObjectURL(blob);
    downloadBlob(url, "pixmatic.svg");
  }
  function downloadBlob(blobUrl, filename) {
    const link = document.createElement("a");
    link.href = blobUrl;
    link.download = filename;
    document.body.appendChild(link);
    link.click();
    document.body.removeChild(link);
    setTimeout(() => URL.revokeObjectURL(blobUrl), 1000);
  }
  function downloadDataURL(dataURL, filename) {
    const link = document.createElement("a");
    link.download = filename;
    link.href = dataURL;
    link.click();
  }
  function saveAsAi(canvasEl) {
    const dataURL = canvasEl.toDataURL("image/png");
    const w = canvasEl.width;
    const h = canvasEl.height;
    const aiContent =
  `%!PS-Adobe-3.0
  %AI-Format 1.2
  %%BoundingBox: 0 0 ${w} ${h}
  %А ось base64 PNG:
  %Base64Begin
```

```
${dataURL}
%Base64End
%EOF
`;
  const blob = new Blob([aiContent], { type: "application/postscript" });
  const url = URL.createObjectURL(blob);
  downloadBlob(url, "pixmatic.ai");
}
```