МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: «Комп'ютерна інженерія»

Група: 2БКС-28

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

здобувача освіти денної форми навчання

БКС.28.22.000.КРБ

САМОЛИГИ
ДМИТРА МАКСИМОВИЧА

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-28**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **«Аналіз ефективності методів покращення цифрових зображень»**

Проектний матеріал складається з пояснювальної записки на 77 сторінках та графічного (презентаційного) матеріалу на 17 аркушах (слайдах)

Виконавець _____ (Самолига Д.М.)

Керівник проекту _____ (Кірєєв І.А.)

Консультанти:

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувач кафедри _____ (Іванова Л.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «26» 06 2024 р. Протокол ЕК № 2

Оцінка ЕК 4 (добре) 85 б

Секретар ЕК _____

АНОТАЦІЯ

У випускній кваліфікаційній роботі виконано реалізацію методів покращення цифрових зображень та аналіз їх ефективності у порівняння з іншими широко розповсюдженими методами масштабування та інтерполяції зображень у комп'ютерній графіці. Розглянуті теоретичні основи масштабування зображень, векторизації, побудови сплайнів, створення графу зв'язності між сусідніми пікселями, побудови і спрощення діаграм Вороного.

Виконано огляд теоретичних основ комп'ютерної графіки, зокрема – для масштабування зображень із низькою роздільною здатністю. Виділено сильні та слабкі сторони кожної групи алгоритмів. Розглянуті та проаналізовані існуючі програмні продукти для покращення якості зображень із низькою роздільною здатністю.

Створено ряд основних функціональних та нефункціональних вимог до алгоритму покращення зображень. Описані основні етапи складання алгоритму для масштабування зображень з низькою роздільною здатністю, який може використовуватись для інтерполяції цифрових зображень.

Обрані програмні засоби та інструменти для розробки, мову програмування, бібліотеку для роботи з растровою графікою та інтегроване середовище розробки.

Реалізовано бібліотеку для покращення зображень із низькою роздільної здатністю при масштабуванні. Реалізовано мовою програмування C++ програмний застосунок для демонстрації можливостей розробленої бібліотеки, який дозволяє відобразити окремі етапи обробки зображення від початкового до вихідного.

Описано заходи з охорони праці та техніки безпеки.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"16" 01 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

здобувачеві освіти *Самолигі Дмитру Максимовичу*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Аналіз ефективності методів покращення цифрових зображень*

затверджена наказом по коледжу від "02" листопада 2023 р. № 244-А2-ОД

2. Термін здачі студентом кваліфікаційної роботи 13.06.2024

3. Вихідні дані до роботи *1. Відомості про алгоритми обробки піксельних зображень та їх параметри; 2. Особливості реалізації алгоритмів для інтерполяції та масштабування зображень; 3. Вихідні файли бібліотек для покращення якості зображень із низькою роздільною здатністю; 4. Реалізувати алгоритм покращення зображення мовою C++ з використанням бібліотеки OpenCV; 5. Порівняти результати роботи реалізованого алгоритму з іншими*

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1. Аналіз існуючих алгоритмів масштабування растрових зображень

2. Короткий огляд існуючих програмних продуктів для масштабування зображень

3. Створення алгоритму покращення зображень та вибір програмних засобів розробки

4. Тестування розробленого застосунку та порівняння ефективності

5. Питання охорони праці та техніки безпеки

5. Перелік графічного матеріалу (слайдів мультимедійної презентації) *Масштабування растрового цифрового зображення; Етапи обробки зображення; Розділення каналів зображення у кольорових форматах; Побудова графу зв'язності пікселів з діагональними перетинами; Евристика ламаної лінії та при визначенні плану і фону та одновалентного вузлу; Діаграми Вороного випадкової множини точок на площині; Формування діаграми Вороного навколо області без діагоналі та з діагоналлю; Узагальнена блок-схема алгоритму обробки зображення; Діаграма взаємодії модулів програми для обробки піксельних зображень; Діаграма класів для реалізованої програми; Порівняння результатів покращення зображень*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний розділ	*Кірєєв І.А.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання 15.01.2024 р.

Керівник роботи *Кірєєв І.А.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Вступ. Аналіз технічного завдання*	8.05.2024	виконав
2.	*Огляд проблеми зображень з низькою роздільною*	11.05.2024	виконав
	здатністю		
3.	*Аналіз систем підвищення якості растрових зображень*	14.05.2024	виконав
4.	*Аналіз вимог до програмного продукту*	16.05.2024	виконав
5.	*Побудова графів зв'язності та діаграм*	19.05.2024	виконав
6.	*Побудова сплайнів та зафарбовування зображень*	22.05.2024	виконав
7.	*Вибір і аналіз програмних засобів розробки*	24.05.2024	виконав
8.	*Розробка структури програмного продукту*	26.05.2024	виконав
9.	*Реалізація візуалізації зображення*	28.05.2024	виконав
10.	*Розробка інструкції для користувача програми*	1.06.2024	виконав
11.	*Тестування програмного забезпечення*	4.06.2024	виконав
12.	*Порівняння результатів роботи програми*	7.06.2024	виконав
13.	*Розробка питань з охорони праці*	9.06.2024	виконав
14.	*Оформлення слайдів мультимедійної презентації*	13.06.2024	виконав

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 28. 22 000. 00 КРБ	Випускна робота		
А4			БКС 28. 22 000. 00 КРБ ПЗ	Пояснювальна записка	1	

Зм.	Арк.	№ докум.	Підпис	Дата	БКС 28. 22 000. 00 КРБ		
Розробив		Самолига Д.М.		9.06	Аналіз ефективності методів покращення цифрових зображень	Літ.	Аркуш 4 / Аркушів 77
Перевірив		Кірєєв І.А.		9.06		Н Д П	
Н. Контр.		Петрашова В.І.		9.06		ВСП "ОТФК ОНТУ" гр.2БКС-28	
Затверд.		Іванова Л.В.		10.06			

ЗМІСТ

ВСТУП

Сучасні монітори здатні відображати мільйони точок. На таких екранах зображення із низькою розділеною можливістю виглядають чи дуже малими, чи неякісними при значному масштабуванні. Це ускладнює адаптацію застарілого програмного забезпечення задля сучасних пристроїв відображення.

Існує чимало підходів задля поліпшення цифрових зображень, але деякі із них можуть збільшити роздільну здатність лише в обмежену число раз (у 2-4 рази), але інші, котрі можуть збільшувати картинку до будь-якого розміру, роблять це не дуже якісно. Як раз крізь це актуальним є здійснення деякого більш «універсального» методу інтерполювання зображень растрового типу. При цьому треба забезпечити достатню швидкість опрацювання методу інтерполювання та економне використання додатком системних ресурсів. Це дозволить, наприклад, масштабувати піксельні ігри у реальному часі чи використовувати метод на мобільних операційних системах, таких як Android.

Задля здійснення покращеного методу інтерполювання зображень треба:

- визначити властивості та принципи існуючих алгоритмів інтерполювання зображень;
- адаптувати один із розглянутих алгоритмів до задачі розтягування довільних растрових зображень;
- реалізувати спосіб зберігання векторного зображення задля візуалізації градієнтних переходів довільної складності;
- створити формат задля уявлення результатів опрацювання методу розтягування;
- визначити відповідні меті та вимогам програмні засоби розробки додатку задля інтерполювання зображень;
- реалізувати додаток задля демонстрації можливостей програми задля інтерполювання зображень.

Таким чином, дана випускна робота пов'язана із аналізом ефективності методів поліпшення цифрових зображень.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Аналіз видів цифрових зображень та їхніх властивостей

1.1.1 Робота із растровими зображеннями

Частина комп'ютерної графіки, що містить справу із обробкою, створенням та зберіганням растрових зображень називається растровою графікою. Обробка графіки даного типу здійснюється растровими графічними редакторами, такими як Microsoft Paint, Adobe PhotoShop, Corel Photo Paint, GIMP та іншими. Растрове зображення представляє собою сітку точок (найменші точки із визначеним кольором), зображених на екрані, папері чи будь-котрих інших пристроях відображення й матеріалах.

Прикладами растрової графіки є фотографії, скани документів. На рис. 1.1. наведено приклад растрового зображення, де видно, що при масштабуванні не вистачає роздільної здатності задля деталізації віддаленого об'єкту на фотознімку.

Рисунок 1.1. Розтягування растрового цифрового зображення

Характеристиками растрового зображення є такі значення:

- число точок по ширині та висоті зображення (наприклад, 1680 × 1050 чи 1024 × 768);
- глибина кольору чи число кольорів (обсяг затраченої пам'яті в бітах задля одного точки);

– колірна модель чи простір зображення – RGB, CMYK, XYZ, YCbCr та ін.

Растрові зображення зберігаються у різних графічних форматах (JPEG, BMP, PNG, GIF та ін.), кожний із котрих застосовує різний підхід задля стиснення та уявлення даних [3].

Загальні переваги растрової графіки є такими:

– можливість створювати та зберігати майже будь-яке зображення, незалежно з складності, на відміну з векторної графіки, де неможливо точно відтворити складні геометричні форми чи ефекти із прозорістю зображення;
– висока швидкість опрацювання будь-котрих зображень (крізь те, що кожний піксел зберігається незалежно);
– задля переважної кількості пристроїв введення-виведення графічної інформації, таких як монітори, принтери, такий формат є стандартним (за винятком, зрозуміло, векторних пристроїв введення-виведення);
– можливо отримувати різні фото-реалістичні ефекти, такі як туман, розмитість, створювати глибину предметів, тонко регулювати кольори.

Однак растрове уявлення графічних даних містить й деякі недоліки, крізь котрі задля зберігання малюнків із нескладними елементами рекомендують використовувати векторну графіку замість, навіть стиснутої, растрової графіки:

– великий розмір файлів у простих зображень (крізь те, що кожний піксел зберігається незалежно);
– неможливість ідеального розтягування картинки: растрове зображення містить точно визначену роздільну здатність й глибину уявлення кольорів, але при збільшенні такого зображення стане втрачатися якість, з’явиться зернистість;
– неможливість друкувати растрові зображення векторними графічними пристроями;
– складність керування окремими фрагментами зображення.

1.1.2 Робота із векторними зображеннями

Уявлення зображення із сукупності геометричних примітивів – точок, ліній, кривих, полігонів, тобто об'єктів, котрі можливо описати математичними виразами, пов'язане із векторною графікою. Її застосовують задля зберігання креслень, шрифтів, рисунків із чіткими контурами. Векторна графіка використовує вектори (набори координат) задля зберігання зображення. Векторні зображення зберігаються у спеціальних векторних форматах (EPS, WMF, SVG, PDF та ін.). Здійснення та редагування векторної графіки здійснюється за поміччю векторних графічних редакторів, таких як Adobe Illustrator, Inkskape, Corel Draw. На рис. 1.2. наведено приклад растрового зображення та векторного зображення символу, де видно, що при масштабуванні векторного зображення спотворення не відбувається й деталізація не зменшується [4].

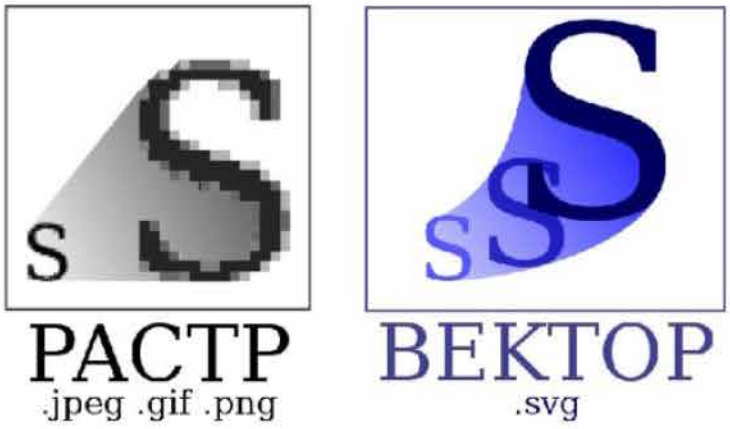


Рисунок 1.2. Зрівняння растрового та векторного цифрового зображення

Дисплеї та принтери є растровими пристроями й перед візуалізацією векторного зображення його треба спочатку перетворити у відповідне растрове уявлення – масив точок. Розмір створеного растрового зображення вже залежить з роздільності пристрою відображення.

Векторне зображення легко перетворити у відповідне растрове, проте зворотне перетворення є дуже складним. Зображення, що було перетворене із векторного формату у растровий займає більший обсяг пам'яті та невідворотно втрачає властивість розтягування без втрати якості. Втрачається й можливість

модифікувати елементи зображення як окремі об'єкти. Розмір векторного зображення залежить з кількості геометричних примітивів, котрими воно представлене, та їхніх властивостей.

Загальні переваги векторної графіки є такими:

- файловий розмір описової частини векторного зображення не залежить з реальної величини зображуваного об'єкта, відповідне велике й мале зображення стане займати однаковий обсяг пам'яті;
- зображення можливо нескінченно збільшувати у зв'язку із тим, що інформація про об'єкт зберігається у описовій формі (простий перелік графічних примітивів та їхніх властивостей);
- можливо легко змінювати параметри зображуваних об'єктів, що надає не погіршувати якість зображення при переміщенні, збільшенні, зменшенні, обертанні навколо осей й т.п.

Однак векторне уявлення графічних даних містить й недоліки, пов'язані із тим, що не все може існувати легко зображене у векторному виді – задля того, щоб зображення було реалістичним та подібним до оригіналу, може знадобитися дуже велика число геометричних примітивів із високою складністю, що негативно вплине на обсяг пам'яті, яку займатиме зображення, та час його візуалізації.

1.2 Застосування піксельної графіки та постановка проблеми

Піксельна графіка (Pixel Art) – це форма зображення, створеного за поміччю растрового графічного редактора, де редагування відбувається на рівні точок, але його роздільна здатність настільки мала, що окремі пікселі чітко видно без розтягування. Тут кожний піксел несе логічне навантаження, й втрата навіть одного повністю зруйнує зображення, на відміну, наприклад, з фотографії, де зміна кольору цілих груп точок може існувати навіть не помітна й не змінює суті зображення. Дуже поширена помилка, що кожний малюнок чи ескіз, зроблений за поміччю растрових редакторів є піксельною графікою. Це невірно, адже «піксельне» зображення відрізняється з «непіксельного» технологією

здійснення, але як раз ручним редагуванням кожного точки зображення. Як раз крізь це стиль Pixel Art відрізняється з інших видів комп'ютерного мистецтва невеликими розмірами, обмеженою колірною палітрою й, як правило, відсутністю згладжування, тобто кожний об'єкт містить чіткі контури. На рис. 1.3 наведено приклад зображення у стилі Pixel Art.

Рисунок 1.3. Приклад зображення у стилі Pixel Art

Pixel Art використовується, зокрема, у комп'ютерних іграх задля старих ігрових приставок, у емуляторах піксельних ігор й багатьох іграх задля смартфонів. Проте, на зображення із низькою розділеною можливістю погано впливає автоматичне розтягування (частіше за все зображення треба перемальовувати), що ускладнює адаптацію старого програмного забезпечення задля сучасних пристроїв виведення.

Програми задля пристроїв, що мали екран із низькою розділеною можливістю, доводиться повністю переробляти задля нормального відображення

на екранах із більшою розділеною можливістю.

Як раз це й визначає проблему використання піксельної графіки у сучасному програмному забезпеченні. Інтерполяція таких зображень за спеціальними алгоритмами, котрі виконувалися б автоматично, дозволила б позбутися цієї проблеми.

1.3 Аналіз існуючих алгоритмів розтягування растрових зображень

Наразі існують три основні методи розтягування зображень – інтерполяція (вираховування проміжних значень), розтягування за поміччю зрівняння кольору точки із його сусідніми пікселами та векторизація.

Далі розглянуто основні алгоритми розтягування піксельної графіки.

1.3.1 Метод Nearest neighbour

Метод «Nearest neighbour (Найближчий сусід)» так само відомий як ступінчаста інтерполяція – метод інтерполювання, при якому у якості проміжного значення вибирається найближче відоме значення функції. У даному випадку просто знаходиться найближчий піксел та використовується його колор (рис.1.4). Інтерполяція методом найближчого сусіда є найпростішим методом інтерполювання. Метод найближчого сусіда зберігає різкі межі, але приносить у зображення ступінчастість (зокрема, діагональні лінії нагадують «сходинки» із квадратів).

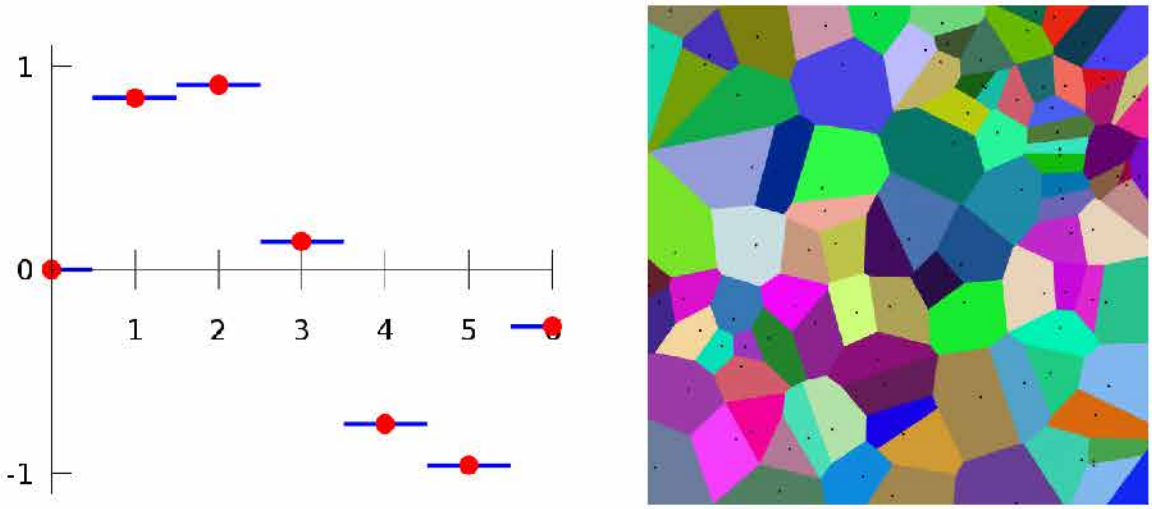


Рисунок 1.4. Інтерполяція за алгоритмом Найближчого сусіда

1.3.2 Метод Eagle

Метод Eagle задля кожного початкового точки генерує 4 вихідних (рис. 1.5). Спочатку кольори всіх чотирьох точок встановлюються в колор поточного початкового точки (C), але далі проглядаються сусідні пікселі. Розглядаються пікселі зверху та зліва: коли вони містять однаковий колор (всі три: верхній, лівий та лівий верхній, тобто S, V, T), то лівий верхній вихідний піксел (1) фарбується в цей колор, інакше колор початкового точки залишається. Аналогічні дії виконуються задля інших трьох вихідних точок (2, 3, 4) й далі задля кожного пікселю початкового зображення. Цей метод масштабує зображення лише у 2 рази, але одиничний чорний піксел при його застосуванні зникає (ця помилка була виправлена в алгоритмах hqnx та 2xSaI).

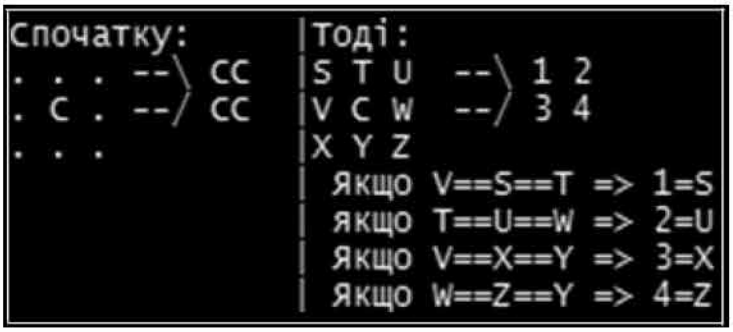


Рисунок 1.5. Принцип опрацювання методу Eagle

1.3.3 Метод Scale2x

Метод Scale2x збільшує зображення лише у 2 рази. Працює наступним чином (рис. 1.6): задля кожного початкового точки (P) формуються 4 вихідні (1, 2, 3, 4) й залежно з кольору сусідів зверху-знизу (A, D) та справа-зліва (B, C) точки (P) встановлюються кольори кожного із 4 вихідних точок.

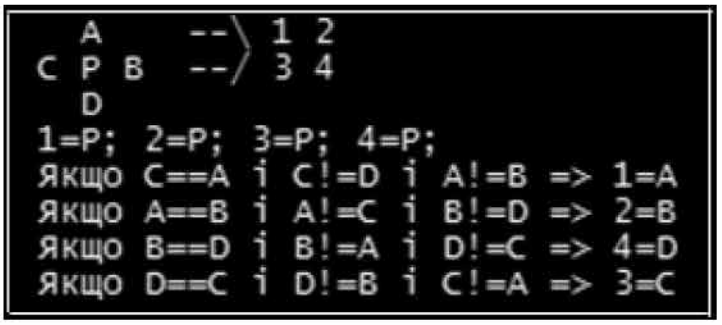


Рисунок 1.6. Принцип опрацювання методу Scale2x

1.3.4 Метод 2xSaI

Метод 2xSaI використовує інтерполяцію та збільшення у 2 рази у порівнянні із алгоритмом Eagle. Спочатку обчислюється середній колір сусідніх точок на вхідному зображенні, але потім цей колір присвоюється новим екстра-пікселям у вихідному зображенні. Отже, цей метод є вдосконаленням методу Eagle (рис.1.7).

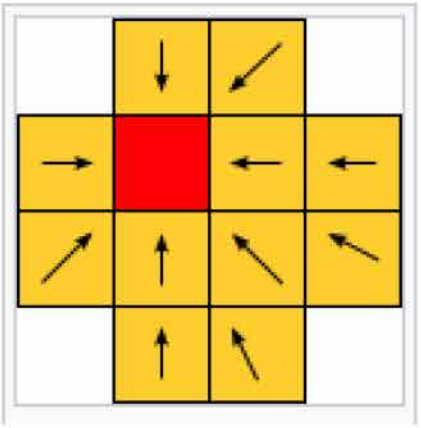


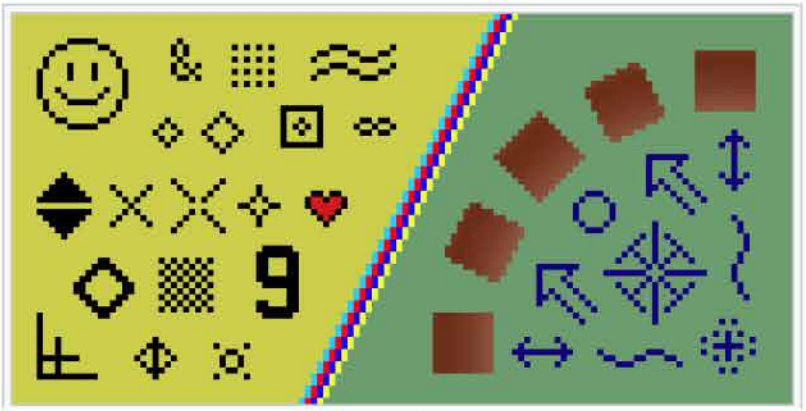
Рисунок 1.7. Матриця оточуючих точок, котрі використовує метод 2xSaI задля розтягування одного точки

1.3.5 Метод hqnx

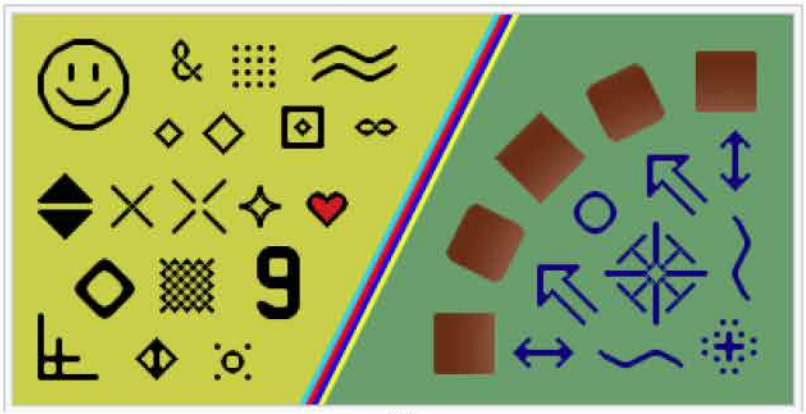
Алгоритми hq2x, hq3x, hq4x збільшують зображення у 2, 3 чи 4 рази відповідно. Колор кожного точки порівнюється із вісьмома сусідніми у колірній моделі YUV (див. розділ 2), сусіди позначаються як близькі й далекі. Далі використовується генерована наново таблиця задля відшукання необхідного співвідношення значень задля кожного із 4, 9 чи 16 вихідних точок.

Метод hq3x відмінно згладжує діагональні лінії із нахилом (кутовим коефіцієнтом) ± 2, ± 1 й ± 0,5 (за умови відсутності згладжування початкового зображення) (рис. 1.8). Круті криві так само добре згладжуються.

На відміну з 2xSaI, у алгоритмах hqnx до вихідного зображення застосовується згладжування, відоме як антиаліасинг – технологія, що використовується при обробці зображень із метою зробити межі кривих ліній більше гладкими, прибираючи зубці, що виникають на краях об'єктів. Завдяки цьому, алгоритми даного типу завоювали популярність.



(але)



(б)

Рисунок 1.8. Зрівняння результатів застосування методу найближчого сусіда (але) та методу hq3x (б)

Недоліками багатьох із розглянутих алгоритмів є можливі невідповідності початкового та вихідного зображень крізь те, що вони порівнюють вхідний піксел лише із сусідніми та збільшують зображення у обмежену число раз (2, 3 чи 4).

1.3.6 Проблема використання існуючих алгоритмів розтягування

Розглянуті методи розтягування були розроблені задля опрацювання природних зображень, задля опрацювання із фотографіями чи художніми малюнками, котрі зазвичай містять високу роздільну здатність. Дані алгоритми засновані на сегментації та виявленні краю об'єктів фільтрами, але такий підхід не стане добре працювати із піксельними зображеннями. Окремі пікселі чи групи точок, котрі можуть існувати ключовою частиною зображення, будуть просто ігноруватися, як шуми й, таким чином, стане отримано недостовірний результат.

1.4 Короткий огляд існуючих програмних продуктів задля розтягування зображень

Задля розтягування зображень із низькою розділеною можливістю існує немало програм. Умовно їхніх можливо поділити на дві групи.

До першої належать library, котрі масштабують зображення в емуляторах та ігрових API. Це такі програмні продукти, як:

- library hq2x, hq3x, hq4x, що використовують метод hqnx. Вихідне зображення виходить досить якісним, добре сприймається оком завдяки застосуванню згладжування, проте містить роздільну здатність більшу лише у 2, 3 чи 4 рази, ніж початкове зображення. Застосувати метод 2 рази підряд щоб збільшити зображення у 16 раз не вийде, адже крізь застосування антиаліасингу при першому масштабуванні виникне розмиття;
- library Bicubic використовує бікубічну інтерполяцію (розширення звичайної кубічної інтерполювання задля функції, що залежить з двох змінних, див. розділ 2). Вихідне зображення виходить суттєво розмивається;
- library Super Eagle використовує метод 2xSaI (удосконалений Eagle) й згладжування вихідного зображення. Зображення у деяких випадках дуже відрізняється з оригіналу й виходить розмитим;
- library Super 2xSaI, за поміччю якої вихідне зображення виходить дещо більш насиченим, ніж у Super Eagle, але багато областей зображення все ж розмиті.

Перелічені library досить непогано масштабують зображення, але збільшують зображення максимум у 4 рази без використання інтерполювання. При збільшенні зображення, наприклад, у 8 чи 16 раз, виникає розмитість [5].

На рис. 1.9 наведені результати опрацювання бібліотек, що належать до першої групи у порівнянні один із іншим.

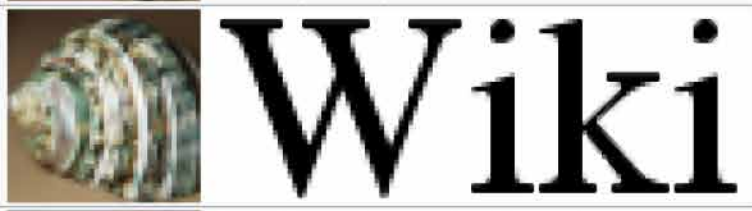
Алгоритм	*Зображення*
Вихідне зображення	
Super-xBR 4x	
Eagle 3x	
hq3x	
Scale 3x	
XBR 3x	
SuperEagle	
SuperSal	
Sal 2x	
Scale 2x	

Рисунок 1.9. Зрівняння результатів розтягування зображення із використанням різних бібліотек

До другої групи відноситься програмне забезпечення, призначене суто задля розтягування зображень. Задля цього найчастіше використовується векторизація. Це такі програмні продукти, як:

- додаток PhotoZoom, у якому використовуються фірмова технологія розтягування S-Spline. Програма є платною й містить закритий програмний код. Зображення масштабується досить добре, проте виходить дещо розмитим;
- додаток CorelDRAW, у якому вхідне растрове зображення перетворюється у векторне й масштабується будь-яку число раз. Ця програма є платною, містить закритий програмний код. Досить часто вихідне зображення значно відрізняється з початкового, втрачаються певні елементи зображення;
- додаток Depixelizer, у якому окрім інструментів задля розтягування міститься так само редактор задля здійснення Pixel Art. Програма призначена задля операційної системи Android й є безкоштовною. Вхідне піксельне зображення перетворюється у векторне й збільшується (максимум у 20 раз). Зображення із малою кількістю кольорів масштабується дуже добре, однак при масштабуванні зображення із трьома чи більше різними кольорами виникають складнощі. Нажаль розтягування займає по часу до кількох десятків секунд, тобто швидкість опрацювання даної програми невисока;
- додатки Adobe Live Trace, Vector Magic, Perfect Resize, Potrace та інші, котрі є платними, містять закритий програмний код, не пристосовані до опрацювання із Pixel Art й масштабують зображення чи не дуже якісно, чи розмивають його.

На рис. 1.10 та 1.11 наведені результати опрацювання деяких із додатків, що належать до другої групи у порівнянні один із іншим. Можливо зазначити, що програми другої групи можуть збільшувати зображення у довільну число раз, проте, жодна із них в результаті чи не відтворює повного змісту початкового зображення, чи не виконує градієнтні переходи й розмиває зображення. Як раз

крізь це є актуальним здійснення іншого підходу задля поліпшення якості зображень із низькою розділеною можливістю.

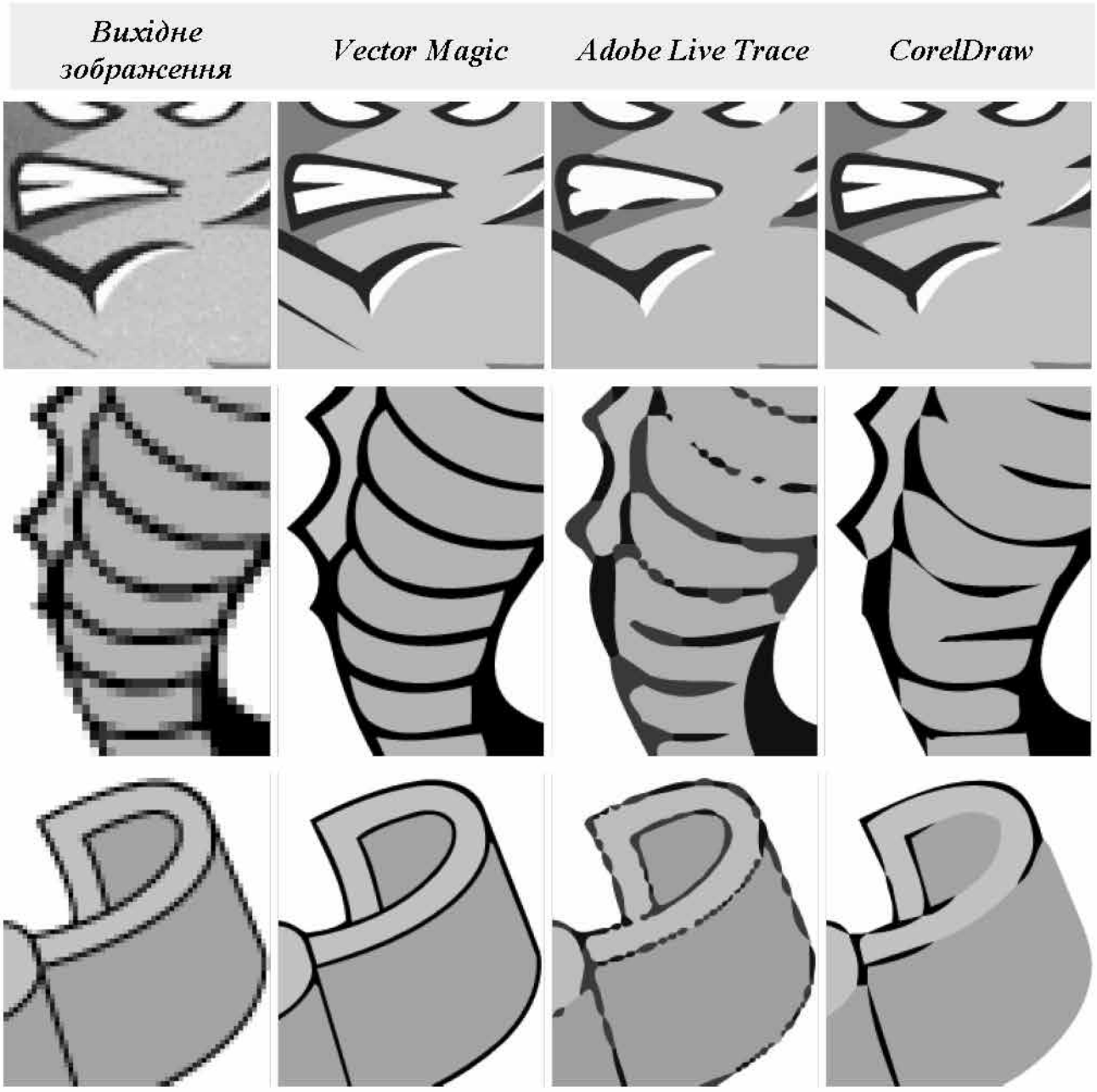


Рисунок 1.10. Зрівняння результатів розтягування зображення із використанням різних програм задля векторизації

1.5 Аналіз вимог до програмних інструментів задля інтерполювання цифрових зображень

На основі аналізу основних переваг, недоліків та функціональних можливостей відомих програмних продуктів задля розтягування растрових зображень, можливо сформувати вимоги до створюваних програмних

інструментів задля інтерполювання цифрових зображень. Треба поєднати сильні сторони існуючих алгоритмів розтягування за поміччю зрівняння кольору точки із його сусідніми пікселами та алгоритмів векторизації зображень й висунути такі вимоги:

1. Програмні засоби містять обробляти піксельну графіку (як розглянуті library hqnx, 2xSaI та інші);
2. Програмні засоби містять масштабувати зображення у довільну число раз (2, 4, 8 й т.д.). При масштабуванні у велику число раз зображення містить не існувати розмитим, крізь це доцільним стане векторизувати зображення;
3. Програмні засоби містять коректно обробляти та реалізовувати градієнтні переходи довільної складності, тобто у результуючому зображенні містить існувати плавний перехід із одного кольору у інший;
4. Програмні засоби містять існувати кросплатформними, швидко обробляти зображення та мінімально використовувати системні ресурси.

При виконанні четвертої умови із'явиться можливість масштабувати піксельні ігри у реальному часі, що не потребуватиме ніяких доробок із боку їхніх розробників.

При виконанні опрацювання растрових зображень піксельної графіки (типу Pixel Art) кожний піксел важливий, крізь це задля початкового й вихідного зображень треба використовувати формати даних, у котрих відбувається стиснення без втрат, зокрема – формат PNG.

Програмні засоби, котрі будуть розроблені у даній роботі, призначені задля розробників програмного забезпечення, зокрема – ігор із піксельною графікою. Як раз крізь це доцільною є реалізація системи у виді library, але не виконуваного файлу, адже вона може існувати легко інтегрована у різні програмні проекти. Задля демонстрації функціональних можливостей розробленої library стане потрібна тестова програма із візуалізацією результатів, отриманих на кожному етапі розтягування зображення. Дане програмне забезпечення доцільно стане створити під ОС Windows.

Метою здійснення програмних інструментів у даній роботі є поліпшення якості зображень із низькою розділеною можливістю при масштабуванні. Як визначено вище, задля цього треба виконати векторизацію вихідного зображення, що дасть можливість інтерполювати його при масштабуванні найбільш якісно. Задля векторного перетворення стане застосовано метод Копфа-Ліщинського, котрий пристосований задля опрацювання із піксельною графікою. Задля здійснення можливості реалізації градієнтних переходів стане використано критерій зрівняння кольору сусідніх точок (котрі містять спільну сторону чи вершину) за алгоритмом hqnx. Це дозволить знаходити схожі за кольором пікселі зображення, котрі сукупно відображають якийсь один логічний елемент піксельного зображення.

Обробка зображення стане складатися із наступних етапів, необхідних задля поліпшення його якості при інтерполюванні:

- етап здійснення дерева когерентності поміж сусідніми пікселами;
- етап здійснення й спрощення декомпозиції Діріхле;
- етап здійснення сплайнів;
- етап розфарбування вихідного зображення.

1.6 Етап здійснення когерентності між сусідніми пікселами

Поміж пікселами у зображенні є ребро, коли вони схожі, але інакше ребра немає. Задля визначення схожості точок вхідне зображення спочатку треба перетворити із колірної моделі RGB у колірну модель YUV. У моделі YUV Y є сигналом яскравості, але U, V – сигналами кольоровості. За формулою (1.1) виконується перетворення до моделі YUV із колірної моделі RGB [6].

$$\begin{cases} Y = K_R \cdot R + (1 - K_R - K_B) \cdot G + K_B \cdot B \\ U = 0.492 \cdot (B - Y) \\ V = 0.877 \cdot (R - Y) \end{cases} \quad (1.1)$$

При цьому значення коефіцієнтів дорівнюють $K_R = 0.299$, $K_B = 0.114$ за рекомендацією BT.601 (чи Rec. 601). Візуальний вигляд компонентів даного

формату при розділенні каналів у порівнянні із відомим форматом RGB показаний на рис. 1.11.

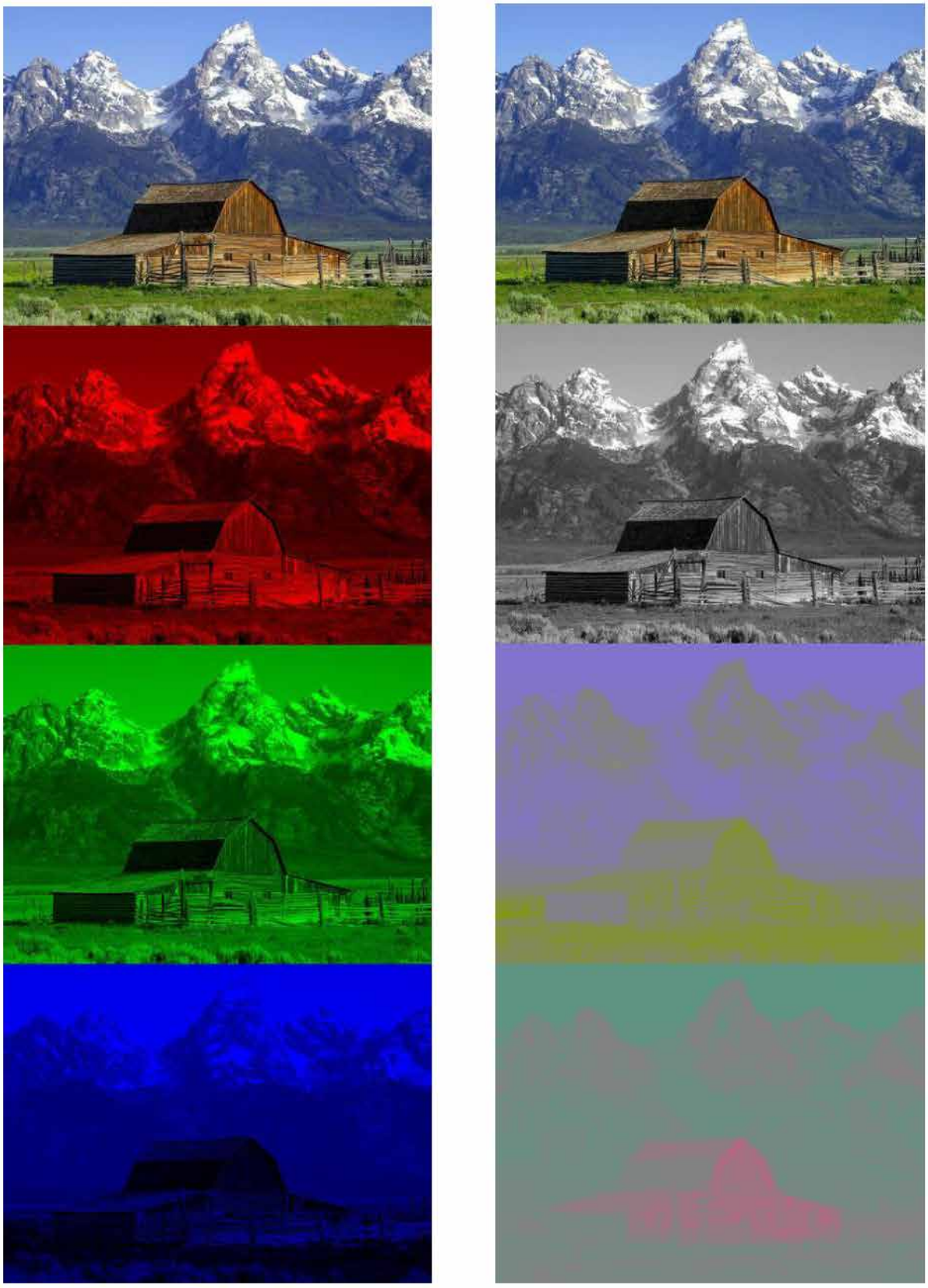
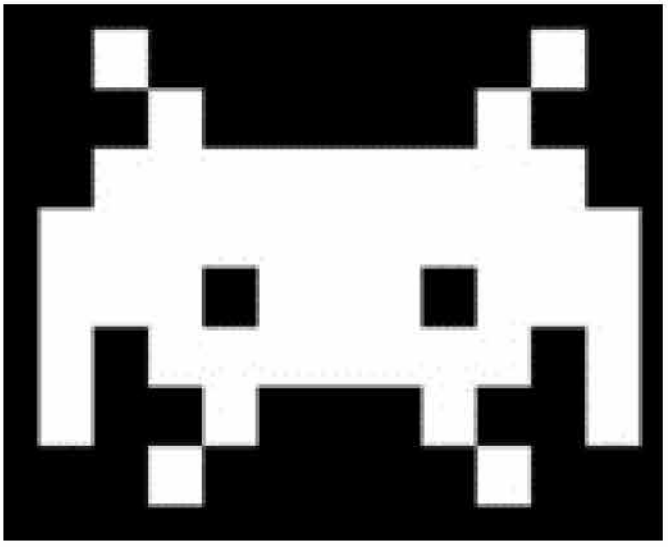
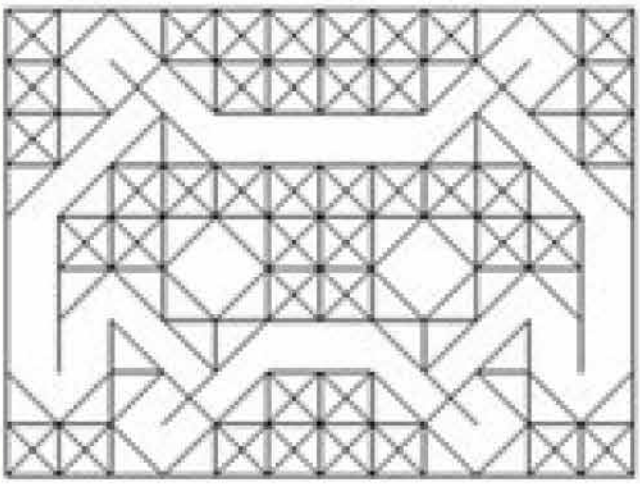


Рисунок 1.11. Розділення каналів зображення у кольорових форматах RGB (зліва) та YUV (справа)

Одразу опісля RGB –> YUV перетворення треба побудувати граф когерентності, у якому кожний вузол – відповідний піксел. Задля цього треба порівняти YUV-канали кожного точки із його сусідами. Коли у сусідніх точок різниця в компонентах *Y, U, V* менша, ніж 48/255, 7/255 й 6/255 відповідно, то це означає, що вони схожі й швидше за все є частиною одного елемента картинки, крізь це вони містять існувати із'єднані. Таким чином встановлюється зв'язок поміж вузлами графа, що відповідають цим пікселям. У іншому випадку пікселі вважаються різними й зв'язку поміж ними не містить існувати. Таким чином стане отримано граф когерентності точок деякого початкового зображення, що показано на рис. 1.12б.

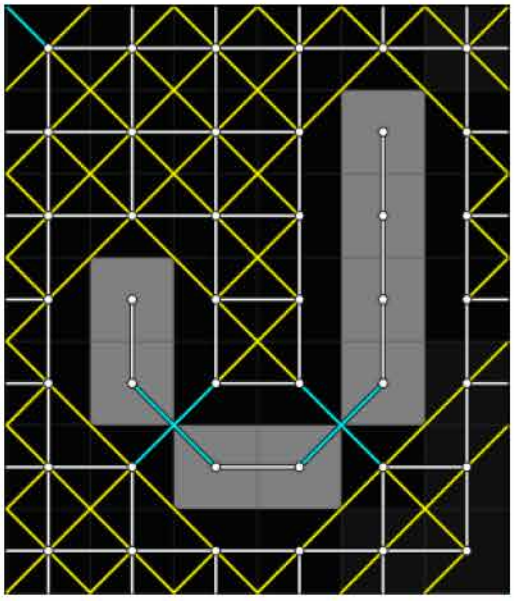


але)

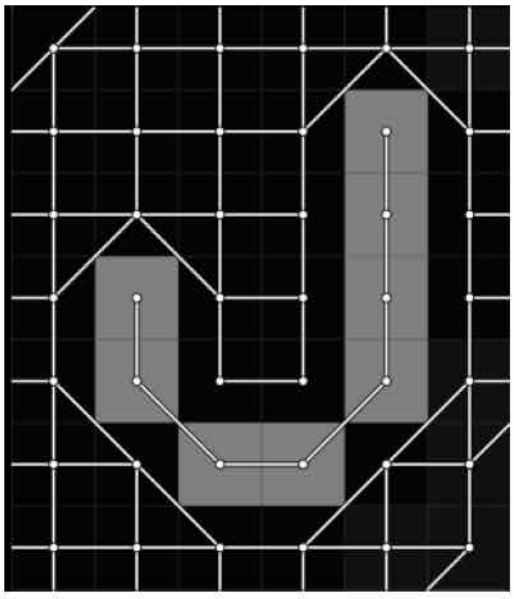


б)

Рисунок 1.12. Побудова дерева точок із діагональними перетинами



але)



б)

Рисунок 1.13. Евристика ламаної лінії із подальшим видаленням діагоналей

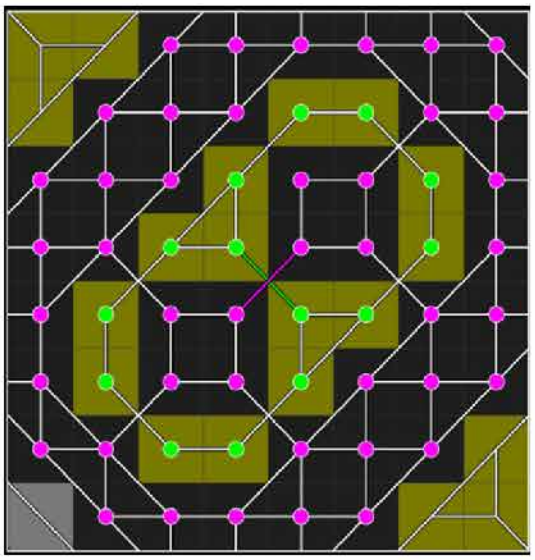
Створений граф когерентності, зазвичай, містить багато похилих перетинів ребр дерева, котрі треба прибрати задля подальшої опрацювання зображення. Задля цього використовуються наступні правила:

- при із'єднанні кожного вузлу дерева із усіма іншими, тобто коли блок 2 × 2 графа повністю сполучений, то він є частиною безперервно заштрихованої області й крізь це можливо видалити два похилих зв'язки, що не вплине на фінальний результат;
- при неповному сполученні блоку 2 × 2 дерева видалення того чи іншого

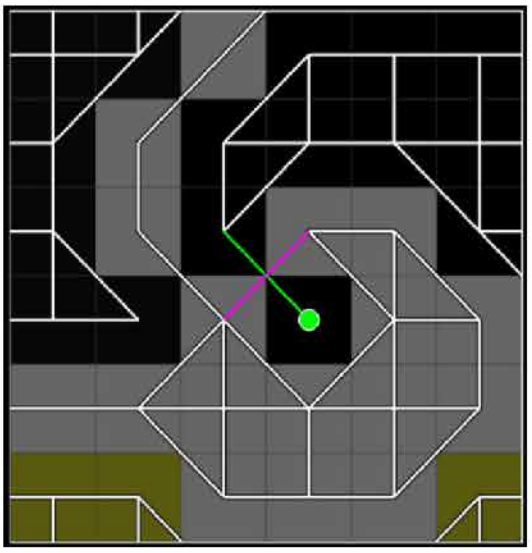
діагонального зв'язку стане по-різному впливати на фінальний результат, тобто треба ретельно вибрати, котрий зв'язок видалити [7].

Задля визначення того, котрий зв'язок треба розривати, треба обчислити три конкретизації задля кожного такого блоку (алгоритми, що можуть вибрати деяке допустиме рішення задачі серед багатьох рішень, але не гарантується, що вибране рішення дійсно стане найкращим):

- коли два точки є частиною довгої ламаної (кривої), як показано на рис. 1.13 (але,б), вони містять існувати із'єднані, адже із високою ймовірністю така крива є логічним елементом зображення. Крива представляє собою послідовність ребр у графі когерентності, котрі зв'язують тільки 2-валентні вузли (вузли, із котрих виходять тільки два зв'язки). Треба обчислити довжини двох кривих, задля котрих кожна діагональ є частиною однієї із них. Коли одна із кривих стане довша, то це означає, що діагональ у блоці, що є частиною цієї кривої, повинна залишитися. Вага даної конкретизації – різниця поміж довжинами двох кривих. В даному випадку чорна ламана містить довжину 7, але біла – лише 1. Як раз крізь це перша крива містить залишитись у результаті;
- колор, якого менше у двоколірних зображеннях, люди сприймають як план, але той, якого більше – як фон. Як раз крізь це пікселі, котрі знаходяться на передньому плані містять існувати із'єднані. Навколо блока 2 × 2 розглядається вікно розміром 8 × 8 й обчислюється число вузлів (можливо крізь інші вузли) в ньому, котрі містять зв'язок із тою чи іншою діагоналлю (рис. 1.14, але). Різниця поміж кількістю вузлів, що містять зв'язок із одною й іншою діагоналлю є вагою конкретизації;
- треба уникати здійснення невеликих окремих однопіксельних зон, тобто треба уникнути фрагментації зображення. Коли одна із двох діагоналей містить 1-валентний вузол (рис. 1.14, б), то краще не розривати її, адже утвориться одиничний піксел. У цьому випадку треба вважати, що вага такої діагоналі дорівнює 5. Різниця поміж вагами двох діагоналей є вагою даної конкретизації.



але)



б)

Рисунок 1.14. Конкретизації при визначенні плану й фону (але) та одновалентного вузлу (б)

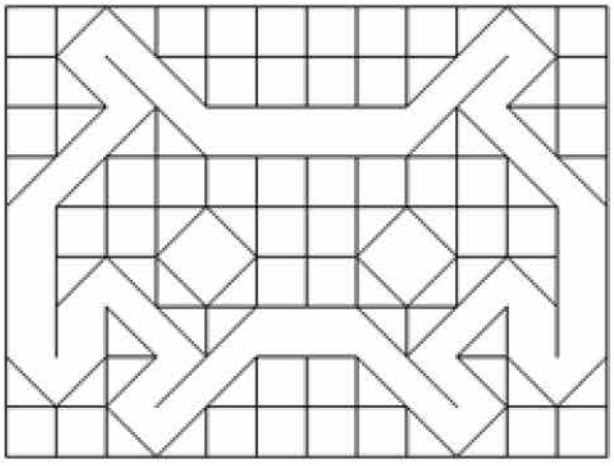


Рисунок 1.15. Граф когерентності точок без похилих перетинів

Треба обрати ту евристику, котра містить найбільшу вагу й розірвати відповідну діагональ у графі когерентності. Коли таких еврістик стане декілька й вони будуть вказувати на те, що треба розривати різні діагоналі, то треба розірвати обидва діагональні зв'язки, опісля чого стане отримано граф без зайвих зв'язків (рис. 1.15).

1.7 Етап здійснення декомпозиції Діріхле та її спрощення

Як відомо, діаграма Діріхле є особливим видом розбиття метричного простору, що визначається відстанями до заданої дискретної множини ізольованих точок цього простору (рис.1.16).

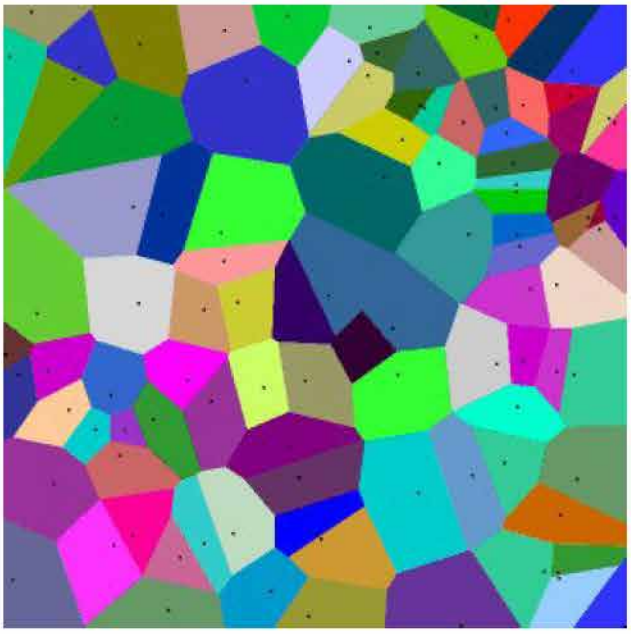
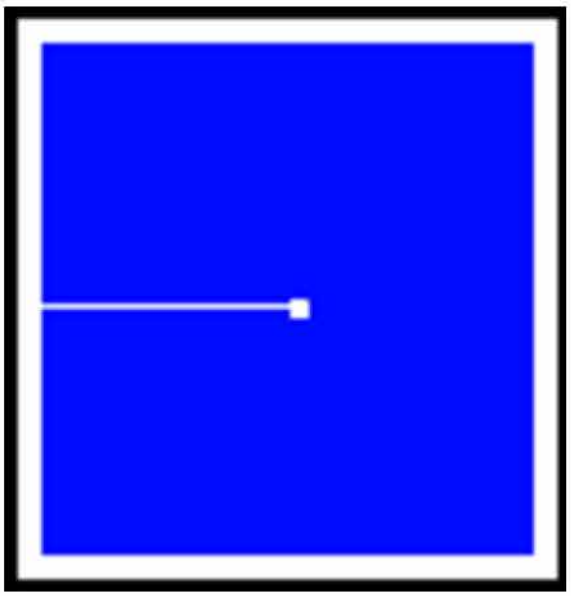


Рисунок 1.16. Декомпозиції Діріхле випадкової множини точок на плоскості

Діаграму Діріхле в узагальненому виді треба побудувати на основі дерева когерентності. Задля цього треба взяти вузли й половинки ребр, котрі виходять із нього, й навколо вузла побудувати область, котра стане мати колор точки, у відповідність якому поставлено цей вузол. Коли взяти будь-яку точку із області, то вона стане лежати ближче до цього вузлу, ніж до інших вузлів чи ребр (чи до півребра, яке виходить із нього).

Розглянемо метод побудови декомпозиції Діріхле. Спочатку навколо кожного вузла треба побудувати квадратну область, котра за розміром така сама,

як розмір точки початкового зображення. Цю область треба змінювати тоді й тільки тоді, коли із вузла виходить діагональне ребро. Змінюється вона завжди однаково в залежності з наявності похилих зв'язків у графі когерентності точок зображення (рис. 1.17).



але)

б)

Рисунок 1.17. Формування декомпозиції Діріхле навколо області без діагоналі (але) та із діагоналлю (б)

ІЗ дерева когерентності (рис. 1.15), таким чином, можливо отримати діаграму Діріхле (рис. 1.18). Тут всі вершини позначено пурпурним кольором, із кожної виходить певна число ребр.

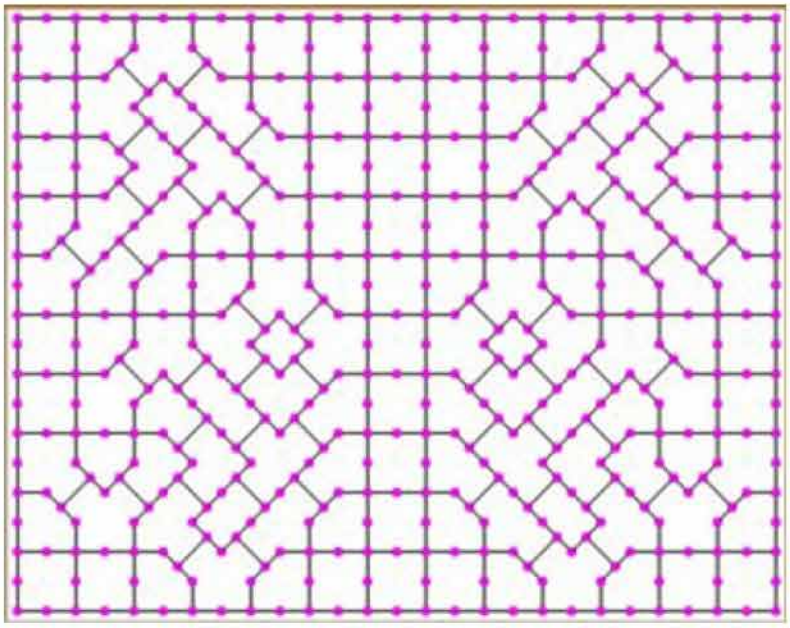


Рисунок 1.18. Побудова декомпозиції Діріхле із дерева

На рис. 1.18 видно, що проявляється обрис зображення. Тепер треба спростити діаграму, видаливши у ній всі двовалентні вершини, тобто ті, із котрих виходить як раз два ребра. Опісля цієї операції діаграма стане максимально схожа на оригінал зображення. Усі вершини, що залишилися у діаграмі, є контрольними точками (на рисунку позначені блакитним кольором). Кожна область (клітина) декомпозиції відповідає певному пікселю у вхідному зображенні (рис. 1.19).

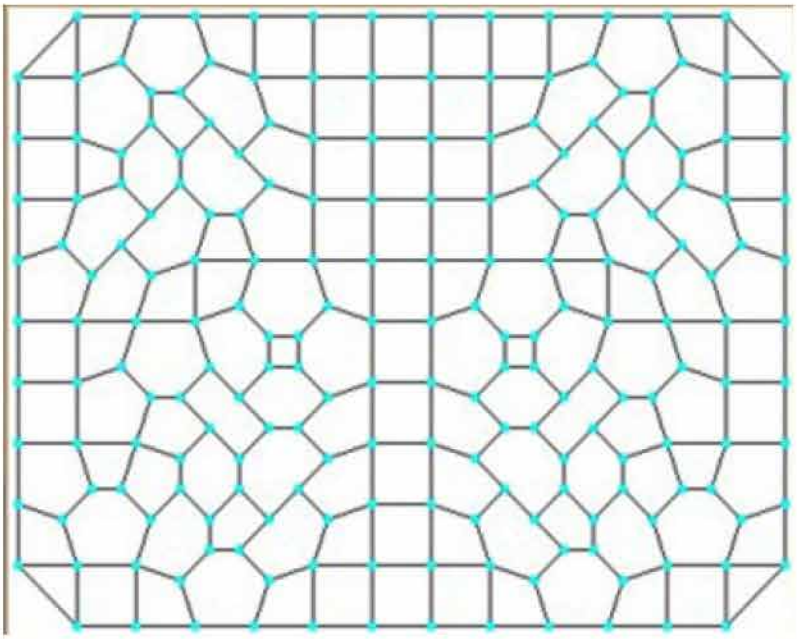


Рисунок 1.19. Побудова спрощеної декомпозиції Діріхле

Опісля цього, за поміччю зрівняння схожості точок початкового зображення, треба знайти схожі області у даній спрощеній діаграмі Діріхле. Аналогічно, коли кольори двох сусідніх областей відрізняються не більше, ніж на 48/255, 7/255 й 6/255 у компонентах *Y, U, V* відповідно, – їхніх можливо сполучити у одну область (тобто видалити у діаграмі ребро поміж ними). Опісля цих операцій стане отриманий обрис зображення (рис. 1.20). На даному етапі зображення вже є сукупністю ламаних ліній.

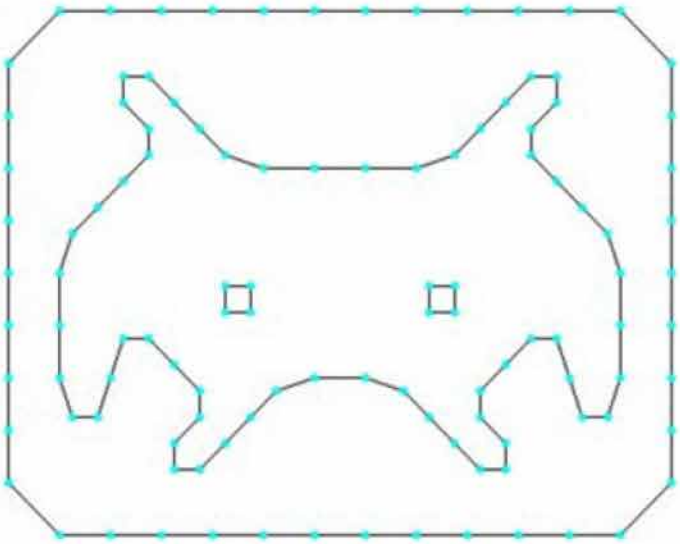


Рисунок 1.20. Обрис зображення із ламаних ліній

1.8 Етап здійснення сплайнів із ламаних ліній

У загальному значенні сплайн є функцією, область визначення якої розбита на шматки, на кожному із котрих процедура представляється деяким поліномом. Сплайн, таким чином, представляє собою універсальну суцільну криву, що складається із декількох простих [8].

Треба відшукати кожну ламану, що входить до контуру зображення. ІЗ цих ламаних далі можливо стане отримати сплайни, котрі описуються математичними формулами. На попередньому етапі вже було отримано векторне уявлення об'єктів, крізь це обрис вихідного зображення вже не стане прив'язаний до масштабу. Треба щоб отримані сплайни проходили крізь усі контрольні точки ламаної, адже це значно спростить подальше розфарбування зображення. Доцільним стане використання кубічного сплайну, адже B-сплайн, котрий хоч й

є візуально більш плавним, не проходить крізь усі точки ламаної, крізь що фінальне розфарбування стане потребувати значно більшої кількості системних ресурсів й стане нетривіальною задачею.

У загальному виді кубічний сплайн є гладкою функцією (містить неперервну похідну), область визначення якої розбита на скінченне число відрізків, на кожному із котрих процедура є деяким кубічними поліномом (многочленом третього ступеня). На рис. 1.21 наведене візуальне уявлення даного сплайну.

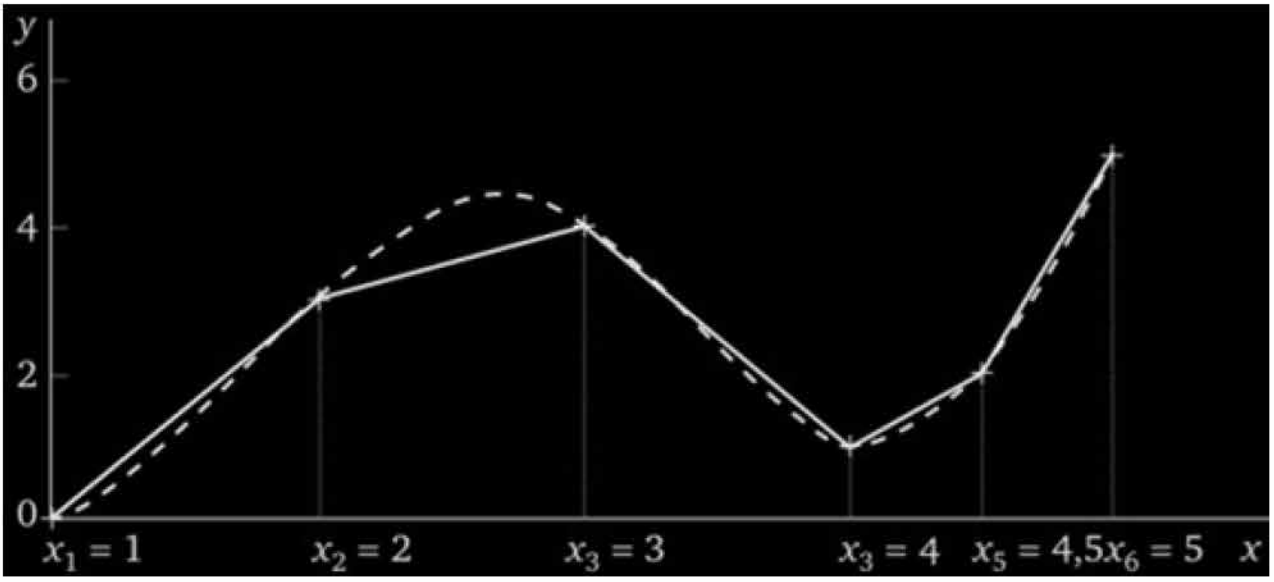


Рисунок 1.21. Здійснення кубічного сплайну задля ламаної лінії

Здійснення сплайну відбувається таким чином. На кожному відрізку виду $[x_{i-1}, x_i]$, де $0 \le й \le N$, процедура $S(x)$ є поліномом третього ступеня $S_i(x)$, коефіцієнти якого треба визначити. При цьому $S_i(x)$ повинна мати неперервні похідні першого й другого порядку задля того, щоб процедура $S(x)$ візуально виглядала гладко на всій області визначення. Так само, за умови, що сплайн проходить крізь усі контрольні точки, містить виконуватись рівність $S_i(x_i) = f(x_i)$. При цьому $S_i(x)$ можливо задля зручності записати у виді:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3 \qquad (1.2)$$

$$S_i(x_i) = a_i, \quad S_i'(x_i) = b_i, \quad S_i''(x_i) = 2c_i, \quad i = \overline{1, N}$$

При виконанні умови неперервності похідних до другого порядку:

$$S_i(x_{i-1}) = S_{i-1}(x_{i-1}),$$
$$S_i'(x_{i-1}) = S_{i-1}'(x_{i-1}), \quad (1.3)$$
$$S_i''(x_{i-1}) = S_{i-1}''(x_{i-1}),$$

Можливо задля зручності записати $h_i = x_i - x_{i-1}$. Тоді формули задля обчислення коефіцієнтів будуть мати вигляд:

$$a_i = f(x_i)$$
$$b_i = \frac{a_i - a_{i-1}}{h_i} + \frac{2 \cdot c_i + c_{i-1}}{3} \cdot h_i \quad (1.4)$$
$$d_i = \frac{c_i - c_{i-1}}{3 \cdot h_i}$$

$$c_{i-1} \cdot h_i + 2 \cdot c_i \cdot (h_i + h_{i+1}) + c_{i+1} \cdot h_{i+1} = 3 \cdot \left(\frac{a_{i+1} - a_i}{h_{i+1}} - \frac{a_i - a_{i-1}}{h_i} \right)$$

Відповідно:

$$c_N = S''(x_N) = 0 \quad (1.5)$$
$$c_1 - 3 \cdot d_1 \cdot h_1 = S''(x_0) = 0$$

ІЗ урахуванням того, що $c_0=c_N=0$, обрахування c_i можливо стане провести за поміччю методу Томаса (прогонки), відомого як метод, котрий застосовується задля вирішення системи лінійних рівнянь виду $Ax = F$, де A є тридіагональною матрицею. Значення c_i будуть знаходитися послідовно, одне за одним. Опісля цього значення усіх інших невідомих у формулі (1.3) стане легко знайти. Такі обчислення треба стане виконати задля кожної ламаної, опісля чого стане отримано масив кривих, що описують обрис зображення. Оскільки кожна крива описується математично формулою, по суті стане отримане векторне уявлення об'єктів зображення, що дозволить масштабувати його у довільну число раз в подальшому [9].

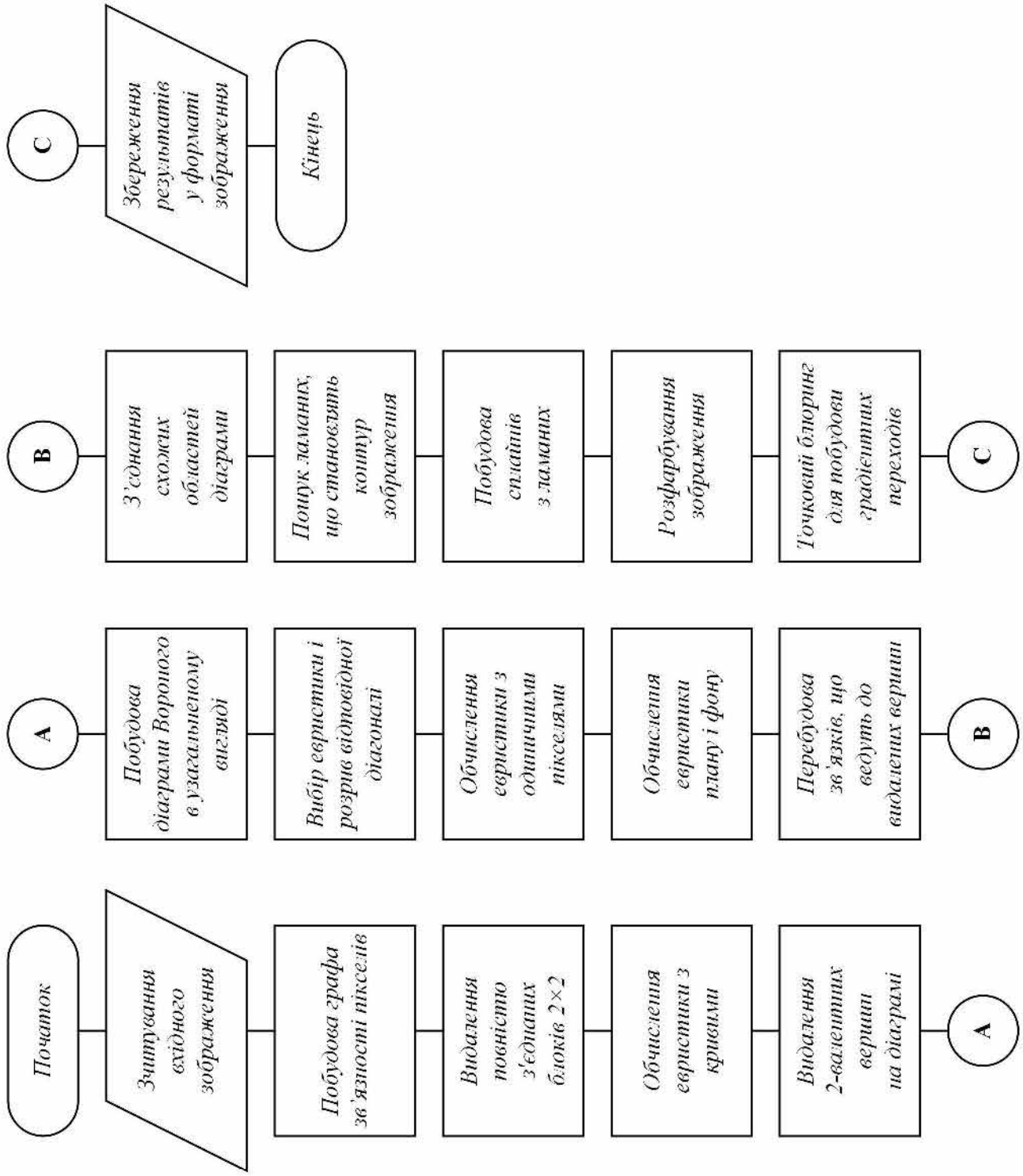


Рисунок 1.22. Узагальнена блок-схема методу опрацювання зображення

1.9 Етап розфарбування вихідного зображення

Опісля виконаних вище етапів треба правильно розмалювати області, що лежать в контурах зображення, враховуючи при цьому розміщення кольорів у

вхідному зображенні.

Кожну область (клітину) спрощеної декомпозиції Діріхле треба зафарбувати в колор, у котрий зафарбований відповідний піксел початкового зображення). Далі, як зазначено вище, коли кольори двох сусідніх областей відрізняються не більше, ніж на 48/255, 7/255 й 6/255 у компонентах *Y, U, V* відповідно, – їхніх можливо сполучити у одну більшу область. При цьому, коли, наприклад, одна клітина мала дещо темніший колор, але інша – дещо світліший, – колор нової сполученої клітини повинен плавно переходити із темнішого у світліший. Такий ефект можливо досягнути за поміччю точкового блюрингу: такий блюринг застосовується не до всього зображення в цілому, але до кожної логічної області окремо. Це гарантує, що фінальне зображення не вийде розмазаним. Опісля виконання розглянутих вище маніпуляцій стане отримане кінцеве зображення – із градієнтними переходами та плавними контурами. Узагальнена блок-схема методу дій по обробці зображення задля поліпшення його якості при інтерполювання, складений на основі описаних вище етапів, наведений на рис. 1.22.

1.10 Вибір й опис програмних інструментів розробки

Вище визначені основні вимоги до розроблюваного програмного забезпечення, але як раз: мінімальне використання системних ресурсів, висока швидкість опрацювання методу та можливість використання створюваної library на різних платформах.

Як видно за розглянутими вище етапами опрацювання зображення, у алгоритмі стане виконуватися достатньо багато математичних дій, зокрема – при побудові сплайнів та операції блюрингу. Враховуючи це, доцільним є використання як раз мови програмування C++, що дозволить досягнути достатньої продуктивності програми при раціональному використанні системних ресурсів [11].

Мова C++, на відміну з мови C, надає використовувати об'єктно-орієнтований підхід при розробці програмного забезпечення. У нових стандартах

(C++17 та C++20) у мову C++ додано багато корисних функцій та класів та покращено продуктивність. До того ж, в подальшому легко можливо стане скомпілювати створювану бібліотеку як статичну (.lib) чи динамічну (.dll). Так само створювана library стане готова задля використання й у виді .hpp (заголовних) та .cpp (початкових) файлів. В подальшому, за потреби, програму можливо стане розпаралелити задля багатоядердних чи багатопроцесорних систем. Мова C++ містить багато стандартних (library thread.h, засоби library windows.h) та сторонніх (бібіліотеки OpenMP та MPI) інструментів задля вирішення даної задачі.

Library OpenCV стане використовуватися задля опрацювання із зображеннями, адже вона є оптимізованою та кросплатформною. У цій бібліотеці реалізовано все, що треба при обробці зображення: зчитування зображень у різних форматах, перетворення із однієї колірної моделі у іншу, збереження результату у різних форматах. Далі детальніше розглянутий кожний із інструментів розробки та відповідні допоміжні засоби.

1.10.1 Можливості мови C++ та необхідний інструментарій

Мова C++ є компільованою, статично-типізованою мовою програмування загального призначення із підтримкою кількох парадигм програмування: узагальненого, процедурного, об'єктно-орієнтованого. Мова C++ широко використовується задля системного програмування, розробки бібліотек, розробки високопродуктивних програм, здійснення драйверів та написання комп'ютерних ігор.

Стандартна library шаблонів (STL) є набором узагальнених алгоритмів, контейнерів, інструментів доступу до їхніх вмісту й різних допоміжних функцій у мові C++. Дана library включена в стандарт C++. Вона значно спрощує розробку програми, дозволяючи створювати менші за обсягом й продуктивніші додатки.

Інтегрованими середовищами розробки (IDE) задля програм на C++ є програмні продукти Microsoft Visual Studio, Embarcadero RAD Studio, JetBrains Clion задля ОС Windows, Linux та macOS.

Інтегроване середовище розробки Microsoft Visual Studio надає розробляти як консольні програми, так й програми із графічним інтерфейсом, в крізь це числі із підтримкою технології Windows Forms, але так само веб-сайти, веб-додатки, веб-служби як в нативному, так й в керованому кодах задля всіх підтримуваних платформ: Microsoft Windows, Windows Mobile, Windows Phone, Windows CE, Microsoft Silverlight, .NET Framework, .NET Compact Framework. Miscrosoft Visual Studio містить редактор із технологією інтелектуального автодоповнення, виконаною на основі машинного навчання. Наявний зручний debugger, редактори форм, дизайнери схем й т.д.

Embarcadero RAD Studio – середовище швидкої розробки додатків фірми Embarcadero Technologies, яке працює під ОС Windows й представляє собою набір інструментів розробки додатків, котрий надає створювати додатки із графічним призначеним задля користувача інтерфейсом задля Windows, Mac OS X, .NET, PHP та веб-рішень. Embarcadero C ++ Builder – це середовище C++, яке повністю відповідає концепції швидкої розробки додатків, об'єднує засоби ANSI C++ й багатофункціональну розширювану інфраструктуру візуальних компонентів.. У C++ Builder 10 із'явився перший у світі компілятор C++ на основі CLANG задля Windows й мобільних платформ із розширеннями RAD PME, що забезпечують швидку розробку задля Windows й інших платформ. Він підтримує тісну інтеграцію із VCL задля Windows й кросплатформними структурами FMX, мова C++ 11 й керування пам'яттю на основі ARC (автоматичного підрахунку посилань) задля C++, але так само містить зворотну сумісність.

Інтегроване середовище розробки JetBrains CLion надає розробляти консольні додатки та library на мові C++. У нових версіях із'явилася підтримка програмно-апаратної архітектури паралельних обчислень CUDA, що надає значно збільшити продуктивність завдяки використанню графічних процесорів. Так само наявні інструменти задля розробки вбудованого програмного забезпечення задля мікроконтролерів STM32. Задля збірки проекту в CLion використовується кросплатформна система CMake, що надає легко інтегрувати library з сторонніх виробників.

1.10.2 Можливості library OpenCV

Library OpenCV містить функції та алгоритми комп'ютерного зору, опрацювання зображень й чисельні алгоритми загального призначення із відкритим кодом. Library надає засоби задля опрацювання й аналізу вмісту зображень, у крізь це числі розпізнавання об'єктів на фотографіях (наприклад, осіб й фігур людей, тексту тощо), вистежування руху об'єктів, застосування методів машинного навчання й виявлення загальних елементів на різних зображеннях, перетворення зображень.

Початковий код library написаний мовою C++ й поширюється під ліцензією BSD. Біндинги підготовлені задля різних мов програмування, таких як Python, Java, Ruby, Matlab, Lua та інших. Може вільно використовуватися у академічних та комерційних цілях. Library містить багату документацію та безліч прикладів, широко застосовується державними структурами, дослідницькими групами та комерційними компаніями, серед котрих Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota та багато інших.

Опісля проведеного огляду можливо зазначити, що із перелічених середовищ розробки найбільш придатним та зручним у роботі стане ICP JetBrains CLion крізь нативну підтримку library OpenCV та загальну поширеність серед розробників програмного забезпечення професійного рівня [12].

У наступному підрозділі випускної опрацювання будуть розроблені програмні засоби задля поліпшення якості цифрових зображень, але як раз – бібліотеку задля розтягування зображень із низькою розділеною можливістю.

Задля демонстрації можливостей розробленої library та зрівняння результатів із іншими алгоритмами стане створена тестова програма, котра дозволить зафіксувати й оцінити результати, отримані на кожному із етапів опрацювання зображення.

1.11 Здійснення об'єктно-орієнтованої моделі

Головний клас програми *Depixelizing* відповідає за поліпшення якості цифрових зображень, тобто реалізує розглянутий вище метод розтягування

зображення із низькою розділеною можливістю шляхом векторизації. Його основні атрибути та методи наведені у табл.1.1.

Таблиця 1.1. Атрибути класу Depixelizing та їхніх опис

Тип	*Назва*	*Опис*
Mat	*m_image*	*Вхідне зображення (матриця точок)*
float	*m_scale*	*Коефіцієнт розтягування*
int	*m_width*	*Ширина початкового зображення*
int	*m_height*	*Висота початкового зображення*
vector <unsigned char>	*m_connections*	*Граф когерентності точок*
list<Vertex>	*m_vertexes*	*Список вершин на діаграмі Діріхле*
list<Edge>	*m_edges*	*Список із'єднань вершин на діаграмі Діріхле*
vector<vector <Vertex>>	*m_curves*	*Список кривих, що утворюють обрис зображення*
vector<vector <list<Vertex>:: iterator>>	*m_pixelsToCells*	*Мапа відповідності областей декомпозиції Діріхле до точок зображення*

Діаграма взаємодії компонентів програми, але як раз – модулів розтягування, вирішення евристик, візуалізації, побудови дерева когерентності, побудови декомпозиції Діріхле та її спрощення, побудови сплайнів, пошуку контурів зображення, показана на рис. 1.23.

Треба зазначити, що типи *vector* та *list* є стандартними у мові C++. *Mat* (скорочення з слова *Matrix*) – стандартний тип зображення у бібліотеці *OpenCV*. Типи *Vertex* та *Edge* – користувацькі, котрі були визначені задля полегшення опрацювання із діаграмою Діріхле. Інтерфейси типів *Vertex* та *Edge* наведені у виді коду C++ на рис. 1.24 та 1.25.

Атрибути класу *Depixelizing*, наведені вище, є приватними й закритими задля кінцевих користувачів library.

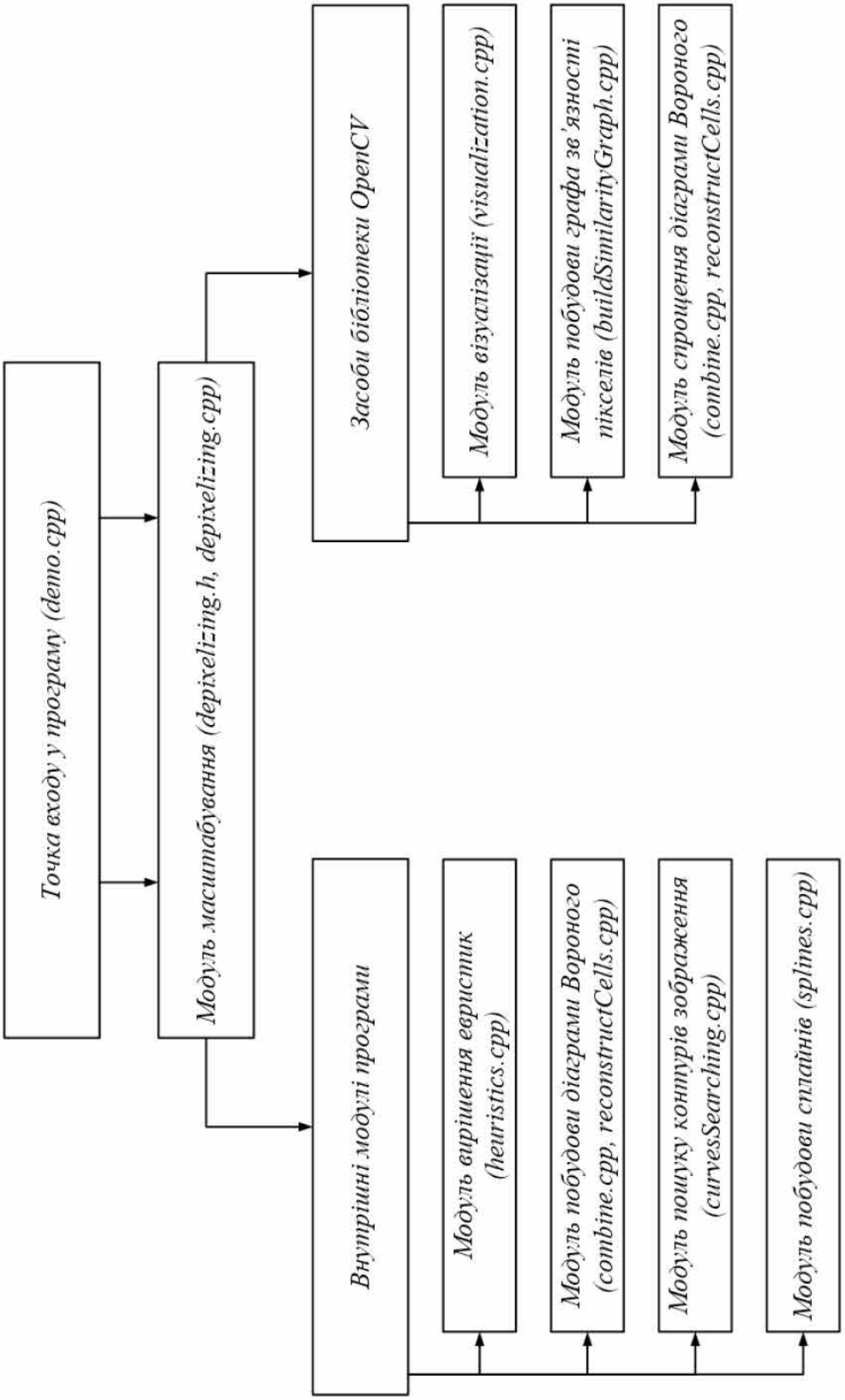


Рисунок 1.23 Діаграма взаємодії модулів програми

```
struct Vertex {
    // координати
    float   x; float y;
    //додаткова змінна для пошуку кривих
    //true - якщо вершина вже відвідана
    bool visited;
    std::list<std::list<Edge>::iterator> edges;
    void addVertex(float x, float y) {
            this->x = x;
            this->y = y;
    }
};
```

Рисунок 1.24. Код на мові C++ задля інтерфейсу типу Vertex

```
struct Edge {
    //вершини на кутi
    std::list<Vertex>::iterator v1;
    std::list<Vertex>::iterator v2;
    //індекси сусідніх пікселів
    // -1 - якщо сусідів немає
    int ind1;
    int ind2;
    void addEdge(int ind1, int ind2, std::list<Vertex>::iterator v1,
    std::list<Vertex>::iterator v2) {
            this->ind1 = ind1;
            this->ind2 = ind2;
            this->v1 = v1;
            this->v2 = v2;
    }
};
```

Рисунок 1.25. Код мовою C++ задля інтерфейсу типу Edge

Користувачам надається метод *depixelize*, котрий приймає два аргументи: вхідне зображення (стандартного типу *Mat* library *OpenCV*) та дійсне число *k* (*float*) – коефіцієнт розтягування.

Результатом функції є зображення (містить тип *Mat*) із покращеною якістю, збільшене у *k* раз. Дане зображення можливо вивести на екран, проводити подальші маніпуляції над ним чи записати у файл. Як раз у даному методі

реалізовано метод розтягування зображення із низькою розділеною можливістю шляхом векторизації, описаний вище. Задля кожного із етапів опрацювання зображення передбачені приватні методи класу *Depixelizing*, котрі послідовно викликаються у публічному користувацькому методі *depixelize*. Основні методи класу *Depixelizing* задля опрацювання зображення наведені у табл. 1.2.

Діаграма класів задля додатку, створена задля опису зв'язків класів *Depixelizing*, *Edge*, *Vertex*, *Mat* та їхніх складових, наведена на рис. 1.26.

Таблиця 1.2. Методи класу Depixelizing

Назва	***Опис***
buildSimilarityGraph	*Побудова дерева когерентності точок*
deleteFullyConnectedBlocks	*Видалення похилих зв'язків у повністю із'єднаних блоках точок 2 на 2*
curvesHeuristic	*Обрахування конкретизації кривих*
sparsePixelsHeuristic	*Обрахування конкретизації плану й фону*
islandsHeuristic	*Обрахування конкретизації із одиничними пікселами*
resolveHeuristics	*Вибір максимальної конкретизації та видалення зв'язків, на котрі вона вказує*
buildCells	*Ініціалізація структур задля зберігання декомпозиції Діріхле в узагальненому виді*
combineCellsDiagonals	*Побудова декомпозиції Діріхле в узагальненому виді*
remove2ValenceNodes	*Спрощення декомпозиції Діріхле (видалення зайвих вершин)*
combineCells	*І3'єднання областей декомпозиції Діріхле*
findAllCurves	*Знаходження ламаних, що складають обрис зображення*
buildSplines	*Побудова сплайнів зі знайдених ламаних*
render	*Фінальне розфарбування зображення, побудова градієнтних переходів*
showSimilarityGraph	*Візуалізація дерева когерентності точок (задля демонстраційної програми)*
showCellGraph	*Візуалізація контурів декомпозиції Діріхле (задля демонстраційної програми)*
showCellImage	*Візуалізація розфарбованої декомпозиції Діріхле (задля демонстраційної програми)*
showSplines	*Візуалізація сплайнів (задля демонстраційної програми)*

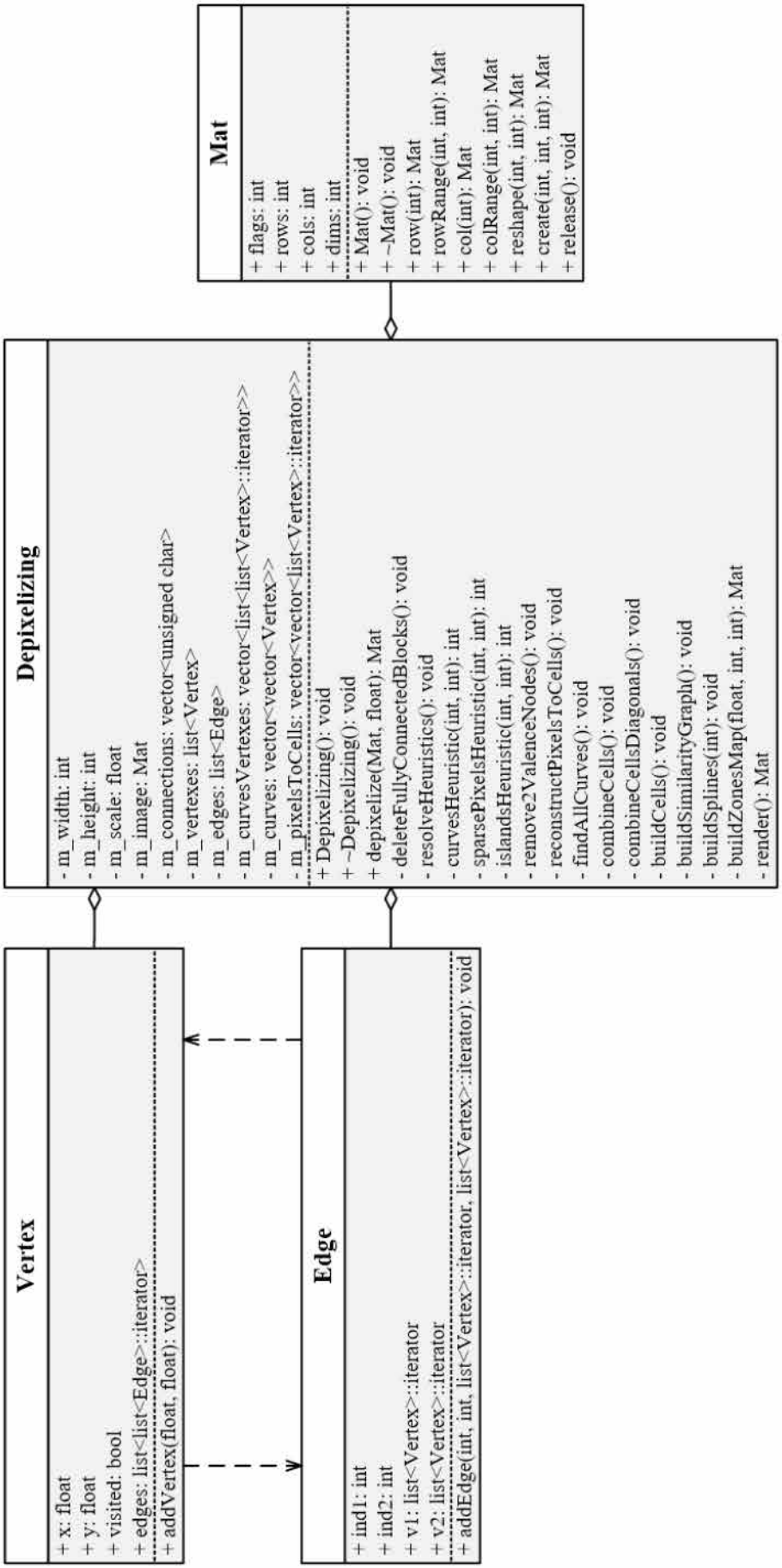


Рисунок 1.26. Діаграма класів задля розробленого додатку

1.12 Програмна реалізація дерева когерентності поміж сусідніми пікселами

Як зазначено вище, задля кожного точки треба зберігати признак його схожості із 8 сусідами: верхнім, нижнім, лівим, правим, верхнім-правим, верхнім-лівим, нижнім-правим, нижнім-лівим. При цьому різниця у компонентах *Y, U, V* менша, ніж 48/255, 7/255 й 6/255 відповідно. Пікселі будуть кодуватися одним байтом, у якому кожний біт відповідає за схожість даних точок.

```
cv::Mat imgYUV(m_image.size(), CV_8UC3);
cv::cvtColor(m_image, imgYUV, cv::COLOR_BGR2YUV);
m_connections.resize(m_width * m_height);
for (int i = 0; i < m_height; ++i) {
    for (int j = 0; j < m_width; ++j) {
        cv::Vec3b& currentPixel = imgYUV.at<cv::Vec3b>(i, j);
        //Верхній
        if (i > 0)
         { cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i - 1, j);
          if (abs(currentPixel[0] - neighbourPixel[0]) < 48 &&
          abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
          abs(currentPixel[2] - neighbourPixel[2]) < 6)
          m_connections[i * m_width + j] |= NEIGHBOUR_TOP;
          }
         //Верхній-правий
        if (j < m_width - 1 && i > 0) {
           cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i - 1, j + 1);
           if (abs(currentPixel[0] - neighbourPixel[0]) < 48 &&
              abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
              abs(currentPixel[2] - neighbourPixel[2]) < 6)
              m_connections[i * m_width + j] |= NEIGHBOUR_TOP_RIGHT;
            }
```

Рисунок 1.27. Код мовою C++ задля побудови дерева когерентності поміж сусідніми пікселами

Коли константа *X* стане входити до суми чисел у вузлі, то у даного точки є сусід, якому відповідає константа *X*. Стане визначено 8 констант, кожна із котрих відповідатиме одному із сусідів:

```
#define NEIGHBOUR_TOP 1
#define NEIGHBOUR_TOP_RIGHT 2
#define NEIGHBOUR_RIGHT 4
#define NEIGHBOUR_BOTTOM_RIGHT 8
#define NEIGHBOUR_BOTTOM 16
#define NEIGHBOUR_BOTTOM_LEFT 32
#define NEIGHBOUR_LEFT 64
#define NEIGHBOUR_TOP_LEFT 128
```

Значення кожного вузлу у графі когерентності на початку дорівнюють 0. Тоді задля кожного точки порівнюється його колор із 8 сусідами та змінюється значення вузлів дерева за поміччю оператора побітового *ЧИ* (|). На рис. 1.27 ці дії показані тільки задля двох сусідів, інші порівнюються по аналогії.

```
for (int i = 0; i < m_height - 1; ++i) {
    for (int j = 0; j < m_width - 1; ++j) {
        if ((m_connections[i * m_width + j] & NEIGHBOUR_RIGHT) &&
            (m_connections[i * m_width + j] & NEIGHBOUR_BOTTOM) &&
            (m_connections[(i + 1) * m_width + j] & NEIGHBOUR_RIGHT) &&
            (m_connections[i * m_width + (j + 1)] & NEIGHBOUR_BOTTOM)
        ) {
                m_connections[i * m_width + j] &= ~NEIGHBOUR_BOTTOM_RIGHT;
                m_connections[(i+1) * m_width + (j+1)] &= ~NEIGHBOUR_TOP_LEFT;
                m_connections[(i + 1) * m_width + j] &= ~NEIGHBOUR_TOP_RIGHT;
                m_connections[i * m_width + (j + 1)] &= ~NEIGHBOUR_BOTTOM_LEFT;
        }
    }
}
```

Рисунок 1.28. Код мовою C++ задля видалення похилих зв'язків у графі

Наприклад, коли у вузлі стане записано число 33 – це означає, що у точки є 2 схожих сусіда: нижній-лівий (*NEIGHBOUR_BOTTOM_LEFT – 32*) та верхній (*NEIGHBOUR_TOP – 1*). Опісля цього треба видалити зайві зв'язки. Задля видалення похилих зв'язків у повністю із'єднаних блоках точок 2 на 2 треба пройти по всіх вузлах та виконати перевірку на одночасне існування двох зв'язків за поміччю побітового оператора *Й* (&). Задля видалення зв'язку треба

виконати побітове *Й* задля поточного вузлу та інвертованого значення константи задля сусіда (інверсія виконується за поміччю побітового оператора *НЕ* (~)). Завдяки використанню побітових операторів вдається досягнути високої швидкості опрацювання програми. Код задля видалення похилих зв'язків у графі представлений на рис. 1.28.

```
unsigned char Depixelizing::valence(unsigned char node)
{
 unsigned char valence = 0;
    while (node)
    {
        valence += node & 1;
        node >>= 1;
    }
    return valence;
}
```

Рисунок 1.29. Код функції мовою C++
задля визначення валентності вузлу дерева когерентності

Конкретизації із кривими та одиничними пікселами потребують допоміжної функції задля визначення валентності вузлу дерева когерентності (кількості зв'язків, що виходять із даного вузлу). Завдяки вибраному способу уявлення даних можливо помітити, що валентність вузлу – це число одиничних бітів у бінарному представленні числа, записаного у даному вузлі. Крізь це треба послідовно пройтися по усім бітам цього числа й підрахувати число одиниць за поміччю побітового *Й* (&) та побітового зсуву вправо (>>). На рис. 1.29. представлено код функції задля визначення валентності вузлу дерева когерентності.

Процедура визначення валентності надає досить просто визначити вузли, котрі є частиною однієї кривої: усі вони повинні існувати із'єднані поміж собою та повинні мати валентність 2 (окрім кінців кривої, що містять валентність 1). Таким чином задля двох вузлів, зв'язок поміж котрими розглядається, треба рекурсивно почати пошук кривої, до якої вони входять, у обидві сторони.

```
std::vector <std::vector <int>> a; // карта зображення
a.resize(m_height);
for (size_t i = 0; i < a.size(); ++i) {
      a[i].resize(m_width);
      memset(&a[i][0], 0, sizeof(int) * m_width): }
a[y][x] = 1;
a[y + 1][x + 1] = 1;
a[y][x + 1] = 2;
a[y + 1][x] = 2;
int count_top_left   =   0;
int count_top_right = 0;
int count_bottom_left = 0;
int count_bottom_right = 0;
//верхній-лівий
int j = x;
int i = y;
bool found = true;
while (found) {
      if (valence(m_connections[i * m_width + j]) == 2) {
          searchNeighbours(i, j, a);
          if (j >= 0 && i >= 0 && j < m_width && i < m_height && a[i][j] == 0)  {
              a[i][j] = 1;
               count_top_left++;
            }
            else found = false;
       }
       else found = false;
 }
```

Рисунок 1.30. Код мовою C++ задля пошуку довжини кривої

На рис. 1.30 наведено код задля пошуку частини кривої, до якої входить верхній-лівий вузол. Аналогічні дії виконуються задля нижнього-правого вузлу, опісля чого треба скласти довжини напів-кривих. Опісля цього аналогічно знаходиться довжина іншої кривої й вага даної конкретизації як різниця поміж довжинами двох кривих. Визначення ваги конкретизації плану й фону відбувається так само рекурсивно: із двох пар похилих вузлів виконується рекурсивний пошук у 4 боки у вікні розміром 8 на 8, та пошук розміру областей, до котрих входять діагональні вузли (рис. 1.31). Різниця поміж розміром

областей є вагою даної конкретизації. Задля визначення ваги конкретизації із одиничними пікселами треба шукати валентість вузлів, що лежать на поточній діагоналі й, коли принаймні один із них містить валентність 1, то привласнювати даній діагоналі вагу 5 (тобто уникати одиничних точок, що призведе до фрагментації фінального результату), інакше – 0 (рис. 1.32). Аналогічні дії виконуються задля іншої діагоналі. Вага цієї конкретизації – різниця поміж вагами обох похилих зв'язків.

```
std::vector <std::vector <int>> a; // карта зображення
a.resize(m_height);
for (size_t i = 0; i < a.size(); ++i) {a[i].resize(m_width);
    memset(&a[i][0], 0, sizeof(int) * m_width);
}
int sx = x;
int sy = y; a[y][x] = 1;
a[y + 1][x + 1] = 1;
a[y][x + 1] = 2;
a[y + 1][x] = 2;
search(y, x, sy, sx, a);
search(y + 1, x, sy + 1, sx, a);search(y, x + 1, sy, sx + 1,
a);
search(y + 1, x + 1, sy + 1, sx + 1, a);
int weight1 = 0;
int weight2 = 0;
for (int i = std::max(0, sy-4); i <= std::min(m_height - 1, sy+4); ++i)
    { for (int j = std::max(0, sx-4); j <= std::min(m_width - 1, sx+4); ++j)
        {  if (a[i][j] == 1)
            ++weight1;
        if (a[i][j] == 2)
            ++weight2; }
}
```

Рисунок 1.31. Код мовою C++ задля пошуку плану й фону у зображенні

ІЗ усіх знайдених евристик обирається та евристика, котра містить найбільше значення ваги по модулю. Таким чином у графі когерентності залишається відповідний діагональний зв'язок, але інший розривається. Знак конкретизації визначає, котрий як раз зв'язок треба розривати: коли вага максимальної конкретизації більша за 0 – розривається діагональ, що йде із

нижнього-лівого вузлу до верхнього-правого, але інакше розривається інша діагональ.

```
int Depixelizing::islandsHeuristic(int x, int y) {
    int weight1 = 0;
    int weight2 = 0;
    if (valence(m_connections[y * m_width + x]) == 1
      || valence(m_connections[(y + 1) * m_width + (x + 1)]) == 1)
        weight1 = 5;
    if (valence(m_connections[(y + 1) * m_width + x]) == 1
      || valence(m_connections[y * m_width + (x + 1)]) == 1)
        weight2 = 5;
    return (weight1 - weight2);
}
```

Рисунок 1.32. Код мовою C++ методу конкретизації одиничних точок

```
void  Depixelizing::resolveHeuristics()  {
   for (int y = 0; y < m_height - 1; y++) {
      for (int x = 0; x < m_width - 1; x++) {
       if ((m_connections[y * m_width + x] & NEIGHBOUR_BOTTOM_RIGHT) &&
         (m_connections[y*m_width+(x+1)] & NEIGHBOUR_BOTTOM_LEFT)) {
           std::vector <int> m(3);
           m[0] = curvesHeuristic(x, y);
           m[1] = sparsePixelsHeuristic(x, y);
           m[2] = islandsHeuristic(x, y);
           int res = INT_MIN;
           int number = 0;
           for (int i = 0; i < 3; i++)
               if (res <= abs(m[i]))
                   res = abs(m[i]);number = i;
           if (m[number] > 0) {
              m_connections[y*m_width+(x+1)] &= ~NEIGHBOUR_BOTTOM_LEFT;
              m_connections[(y+1)*m_width+x] &= ~NEIGHBOUR_TOP_RIGHT; }
           else { m_connections[y*m_width+x] &= ~NEIGHBOUR_BOTTOM_RIGHT;
                  m_connections[(y+1)*m_width+(x+1)] &= ~NEIGHBOUR_TOP_LEFT; }
            }
        }
   }
}
```

Рисунок 1.33. Код мовою C++ методу евристик похилих перетинів

Опісля вирішення евристик задля усіх похилих перетинів (рис. 1.33) стане отримана фінальна версія дерева когерентності, на основі якої можливо стане будувати діаграму Діріхле в узагальненому виді та проводити подальшу обробку зображення.

1.13 Програмна реалізація декомпозиції Діріхле

Під час програмної реалізації декомпозиції Діріхле будуть використовуватися вершини *Vertex* та зв'язки поміж вершинами *Edge*, котрі були описані на рис. 1.24 та 1.25. Таким чином кожний піксел перетворюється у вісім вершин та зв'язки поміж ними. Виконуючи аналіз дерева когерентності, при існуванні діагонального зв'язку на ньому, треба змінити діаграму так, як показано на рис. 1.17б. Задля цього треба додавати нові вершини та рухати існуючі (рис. 1.34). ІЗ отриманої декомпозиції Діріхле треба видалити усі 2-валентні вершини та зв'язки, що до них ведуть, й отримати спрощену діаграму Діріхле (рис. 1.35).

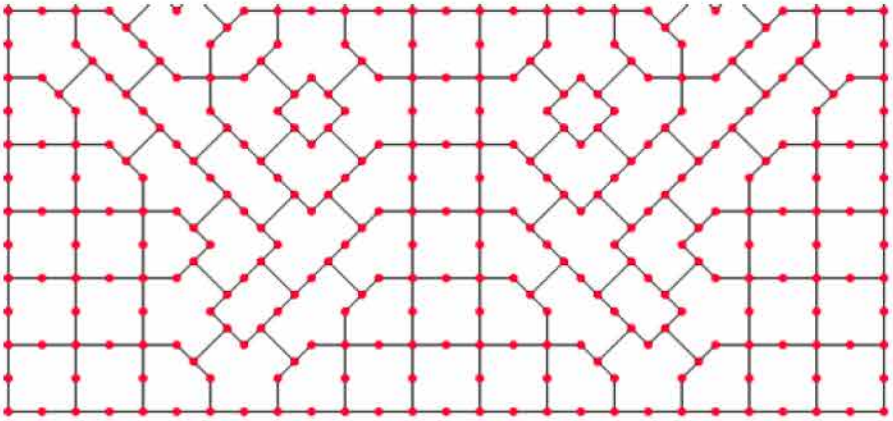


Рисунок 1.34. Додавання нових вершин та зсув існуючих у діаграмі Діріхле

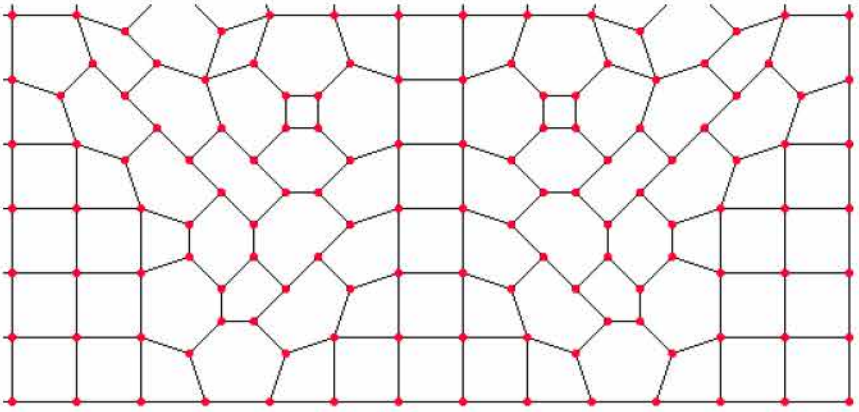


Рисунок 1.35. Отримання декомпозиції Діріхле спрощеного типу

```
void Depixelizing::combineCells() {
    auto it = m_edges.begin();
    while (it != m_edges.end())
    { auto currIt = it;
      ++it;
        if (connected(currIt->ind1, currIt->ind2))
            {currIt->v1->edges.remove(currIt);
            currIt->v2->edges.remove(currIt);
            m_edges.erase(currIt);
        }
    }
}
```

Рисунок 1.36. Код мовою C++ методу задля із'єднання схожих областей у діаграмі Діріхле спрощеного типу

Опісля отримання декомпозиції Діріхле спрощеного типу треба із'єднати області, що містять схожий колор. Задля цього треба пройти по всіх зв'язках (*Edge*) та видалити ті, котрі є зайвими (рис. 1.36). Опісля виконання вищенаведеного методу стане отримано набір ламаних *m_curvesVertexes*, котрі становлять обрис вихідного зображення (рис. 1.37). Задля досягнення плавних переходів далі будуть побудовані сплайни із даних ламаних.

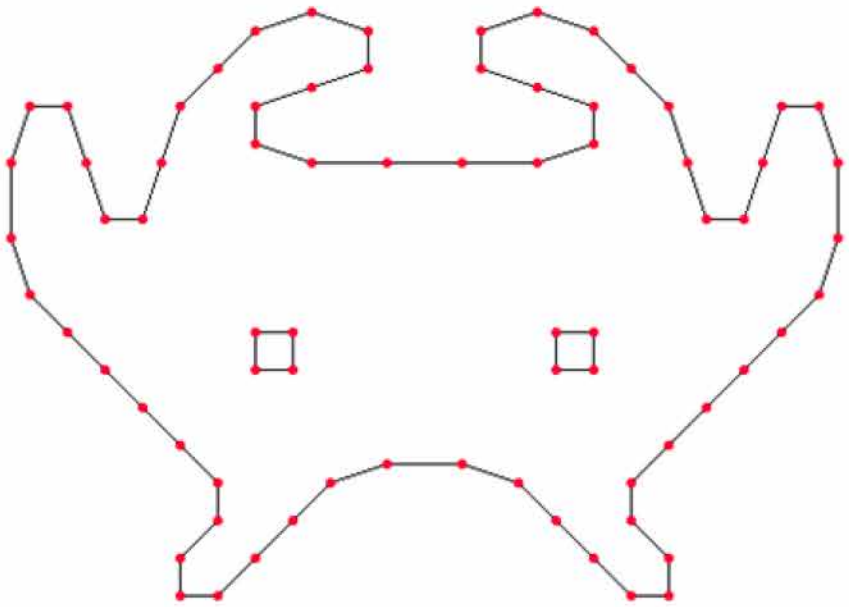


Рисунок 1.37. Ламані, що складають обрис зображення

1.14 Програмна реалізація сплайнів задля згладжування

На попередньому етапі отримано сукупність ламаних, що становлять обрис зображення. Опісля цього треба задля них побудувати сплайни, на котрих можливо стане обрати достатньо контрольних вершин, щоб обрис здавався плавним. Таким чином, треба створити додаткові вершини на кожній ламаній, причому число вершин залежить з коефіцієнту розтягування. Чим більший коефіцієнт розтягування, тим більше вершин треба задля уникнення зубчастості контуру фінального зображення.

Оскільки використовується кубічний сплайн, – задля знаходження його коефіцієнтів на певній області треба чотири контрольні точки (адже кожний многочлен ступеня N однозначно визначається крізь $(N + 1)$ точок). Таким чином задля кожної точки ламаної треба знайти чотири сусідні контрольні точки й побудувати задля неї *steps* додаткових вершин (рис 1.38).

```
for (int j = 0; j <= steps; ++j)
   { float t = (float)j / (float)steps;
  float k1 = 2 * t * t * t - 3 * t * t + 1;
  float k2 = t * t * t - 2 * t * t + t;
  float k3 = -2 * t * t * t + 3 * t * t;
  float k4 = t * t * t - t * t;
  Vertex v;
  v.x = (k1 * controlPoints[1].x + k2 * controlPoints[0].x+
  k3 * controlPoints[2].x + k4 * controlPoints[3].x);
  v.y = (k1 * controlPoints[1].y + k2 * controlPoints[0].y+
     k3 * controlPoints[2].y + k4 * controlPoints[3].y);
  m_curves[i].push_back(v);
    }
```

Рисунок 1.38. Код мовою C++ побудови сплайнів задля згладжування

Експериментальним шляхом встановлено, що найкращі результати досягаються, коли значення *steps* дорівнює коефіцієнту розтягування. Таким чином при збільшенні зображення, наприклад, у 40 раз, задля кожної точки ламаної, що описує обрис зображення, треба добудувати 40 додаткових вершин.

Опісля виконання описаних дій стане отримано набір кривих *m_curves*, що

описують обрис зображення (рис. 1.39).

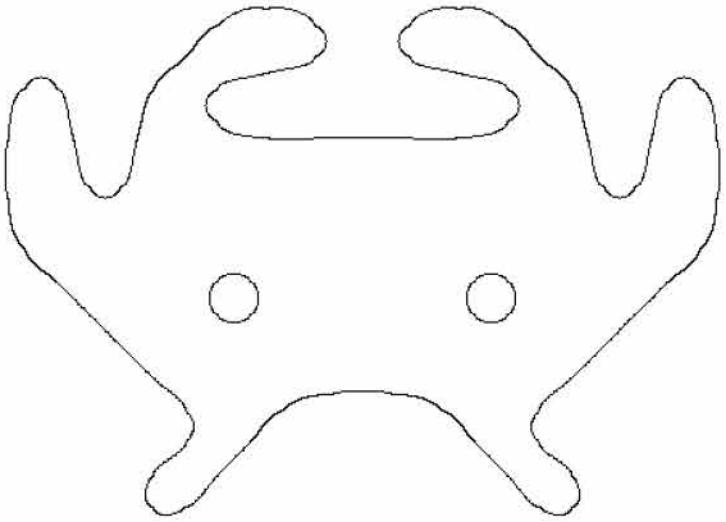


Рисунок 1.39. Набір кривих, що описують обрис зображення

1.15 Програмна реалізація візуалізації вихідного та проміжних зображень

Вихідне та проміжні зображення візуалізуються за поміччю стандартних інструментів library *OpenCV* у ICP JetBrains CLion.

```
void Depixelizing::showCellGraph() {
    const int scale = (int)m_scale;
    cv::Mat img(m_height * scale + 10, m_width * scale + 10, CV_8UC3,cv::Scalar(0,0,0));
    for (auto it = m_edges.begin(); it != m_edges.end(); ++it)
      {
       cv::line(img, cv::Point(cvRound(it->v1->x * scale + 5), cvRound(it->v1->y * scale + 5)),
       cv::Point(cvRound(it->v2->x * scale + 5), cvRound(it->v2->y * scale + 5)),
       cv::Scalar(255, 255, 255), 1);
       cv::circle( img, cv::Point(cvRound(it->v1->x * scale + 5),
       cvRound(it->v1->y * scale + 5)), 1, cv::Scalar(200, 255, 0), 2);
       cv::circle( img, cv::Point(cvRound(it->v2->x * scale + 5),
       cvRound(it->v2->y * scale + 5)), 1, cv::Scalar(200, 255, 0), 2);
       }
    cv::imshow("CellGraph", img); cv::waitKey(0);
}
```

Рисунок 1.40. Код мовою C++ методу задля візуалізації зображення

Таблиця 1.3. Функції й типи у бібліотеці OpenCV та їхніх опис

Назва	*Опис*
Mat	*Матриця точок (контейнер, що зберігає зображення). При ініціалізації приймає цілі висоту, ширину та початковий колор (Scalar)*
Point	*Тип задля зберігання координат зображення, при ініціалізації приймає координати X (int) та Y (int)*
Scalar	*Тип задля зберігання даних точки (кольору), набір довільної кількості дійсних чисел (double)*
line	*Процедура задля побудови лінії, приймає 2 точки (Point), колор (Scalar) та товщину (int)*
circle	*Процедура задля побудови кола, приймає точку центра (Point), радіус (int) колор (Scalar) та товщину (int)*
fillPoly	*Процедура задля зафарбування полігону зображення, приймає список вершин полігону (Point) та колор (Scalar)*
imread	*Процедура задля зчитування зображення приймає ім'я файлу, повертає зображення у форматі Mat*
imshow	*Процедура задля виведення зображення на екран, приймає ім'я вікна (string) та зображення (Mat)*
imwrite	*Процедура задля запису зображення у файл, приймає ім'я файлу (string) та зображення (Mat)*
resize	*Процедура задля найпростіших методів розтягування зображення (nearest neighbour, interpolation), приймає вхідне й вихідне зображення (Mat), коефіцієнти розтягування по осях X й Y та метод розтягування*
waitKey	*Допоміжна процедура, призупиняє виконання програми до натиснення якоїсь клавіші*

На рис. 1.40 наведено програмний код задля візуалізації декомпозиції Діріхле, зображеної на рис. 1.34, 1.35 та 1.37. Виконується прохід по усім зв'язкам поміж вершинами (*Edge*) та будування на кінцях за поміччю функції *circle* двох кіл та за поміччю функції *line* лінії поміж ними.

1.16 Використання створеної library задля поліпшення піксельних зображень

Створена library задля інтерполювання при масштабуванні зображень із низькою розділеною можливістю може використовуватись у будь-котрих програмних продуктах задля опрацювання із растровою піксельною графікою. Користувачу надається метод *depixelize* класу *Depizelizing*, котрий приймає

вхідне зображення й коефіцієнт розтягування, та повертає вихідне зображення покращеної якості. На рис. 1.41 наведений приклад програми із використанням створеної library. Library *depizelizing.h* та *OpenCV* імпортуються, опісля чого виконується зчитування початкового зображення. Опісля цього створюється екземпляр *pix* класу *Depixelizing*, котрий стане виконувати розтягування та поліпшення якості зображення, викликається метод *depixelize* задля зображення *img* та коефіцієнту розтягування *40*. Вихідне покращене зображення записується у змінну *res* й виводиться у файл *output.png*.

```
#include "depixelizing.h"
#include <opencv2/opencv.hpp>
int main() {
    cv::Mat img = cv::imread("input.png");
    Depixelizing pix;
    cv::Mat res = pix.depixelize(img, 40);
    cv::imwrite("output.png", res);
    return 0;
}
```

Рисунок 1.41. Код мовою C++ прикладу використання створеної library

1.17 Тестування застосунку задля демонстрації методу

Додаток задля демонстрації опрацювання створеного методу задля інтерполювання цифрових зображень можливо запустити за поміччю інтегрованого середовища розробки Clion чи напряму крізь утиліту *Cmake*. Задля цього треба у папці із програмним кодом виконати команди, зображені на рис. 1.42.

```
/PixelArt$ cmake --build cmake-build-release --target PixelArt
Scanning dependencies of target PixelArt
[  9%] Building CXX object CMakeFiles/PixelArt.dir/demo.cpp.o
[ 18%] Linking CXX executable PixelArt
[100%] Built target PixelArt
/PixelArt$ cd cmake-build-release/
/PixelArt/cmake-build-release$ ./PixelArt
Enter pixel art file name: inputs/12.png
Enter scale factor: 20
```

Рисунок 1.42. Інтерфейс додатку демонстрації опрацювання створеного методу

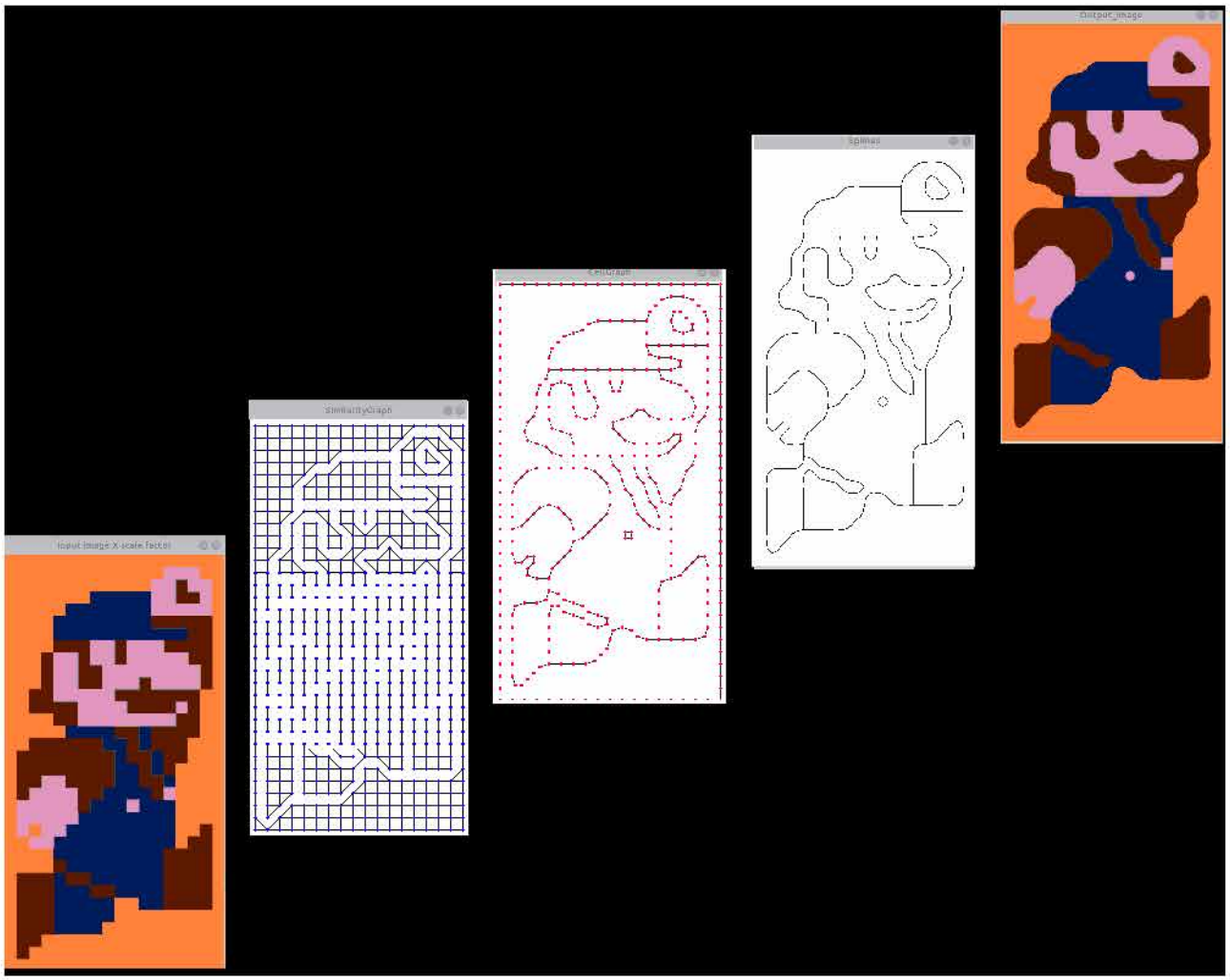


Рисунок 1.43. Покрокове відображення результатів опрацювання методу інтерполювання та розтягування зображень

Задля демонстрації опрацювання створеного методу був розроблений консольний інтерфейс, котрий запитує у користувача ім'я початкового файлу та коефіцієнт розтягування. Додаток покроково відобразить на екрані усі етапи опрацювання зображення (рис. 1.43). Вхідне зображення задля даного прикладу містить роздільну здатність 18 на 34, але вихідне зображення збільшене у 20 раз й містить роздільну здатність 360 на 680. Задля переходу до наступного кроку користувачу треба натиснути будь-яку клавішу клавіатури. Остаточний результат стане записаний у файл та відображений на екрані.

1.18 Зрівняння результатів опрацювання із аналогами

Задля перевірки ефективності створених програмних інструментів задля інтерполювання зображень виконано тестування на декількох прикладах [13]. Результати поліпшення якості цих зображень у порівнянні із результатами, отриманими із інших програм (див розділ 1), наведені на рис. 1.44-1.49.

але) початкове *б) у Adobe Live Trace* *в) за створеним алгоритмом*

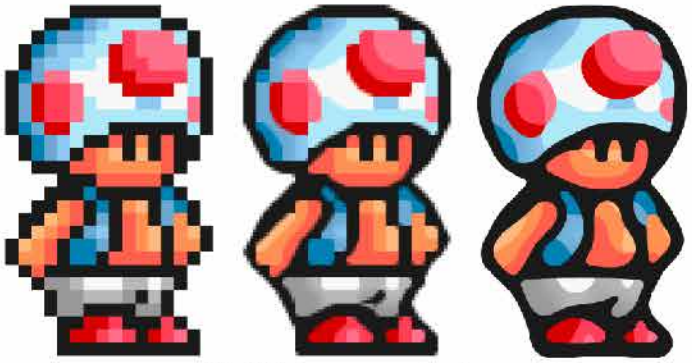
Рисунок 1.44. Результати зрівняння зображення 1

але) початкове *б) у Photo Zoom 4* *в) за створеним алгоритмом*

Рисунок 1.45. Результати зрівняння зображення 2

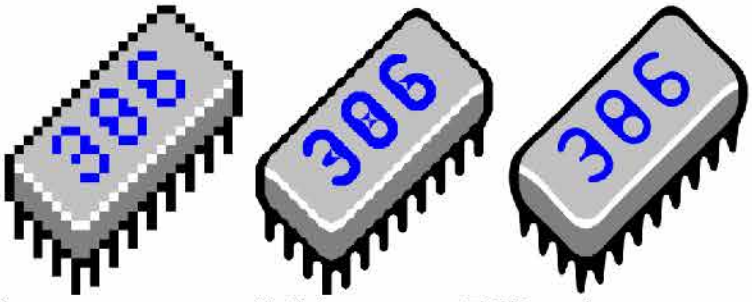
але) початкове *б) бібліотекою hq4x* *в) за створеним алгоритмом*

Рисунок 1.46. Результати зрівняння зображення 3



але) початкове *б) бібліотекою hq4x* *в) за створеним алгоритмом*

Рисунок 1.47. Результати зрівняння зображення 4

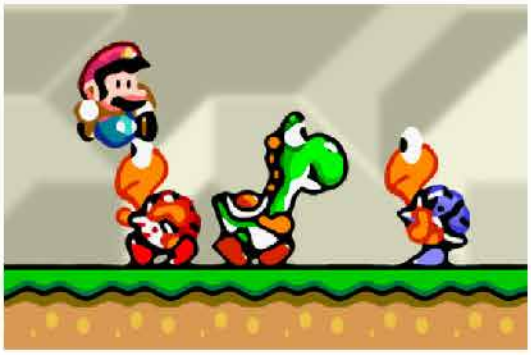


але) початкове *б) бібліотекою EPX* *в) за створеним алгоритмом*

Рисунок 1.48. Результати зрівняння зображення 5

але) початкове

б) бібліотекою hq4x



в) за створеним алгоритмом

Рисунок 1.49. Результати зрівняння зображення 6

2 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

У галузі охорони праці є Закон України «Про охорону праці», дія якого поширюється на всі підприємства, установи й організації незалежно з форм власності та видів їхніх діяльності, на усіх громадян, котрі працюють на цих підприємствах.

Власник чи уповноважені ним органи зобов'язані дбати про умови праці працівників, полегшувати їхніх, оздоровляти навколишнє середовище, дбати про виконання правил безпеки й інструкцій по техніці безпеки.

Координує всю цю діяльність служба охорони праці, котра в залежності з чисельності працюючих може функціонувати як самостійний структурний підрозділ (число працюючих 50 й більше), чи у виді групи спеціалістів чи одного спеціаліста, у крізь це числі за сумісництвом (число працюючих 20 й менше). Задачі службі охорони праці та її функції викладені в Типовому положенні про службу охорони праці».

Працівники так само повинні відповідально ставитись до охорони праці, знати та виконувати вимоги, визначені нормативною документацією. В сучасних умовах кожному працівнику треба постійно підтримувати високий фізичний, психологічний та фаховий рівень, запобігати виникненню випадків травматизму та профзахворювань.

2.1 Аналіз небезпечних та шкідливих чинників, що впливають на працівника

Безпечні умови праці на підприємстві досягаються за рахунок забезпечення безпеки виробничих процесів, котрі обґрунтовані й прийняті в технологічній частині дипломного проекту.

Задля установлення можливого впливу на здоров'я користувачів ВДТ виробничих чинників містить значення ряд якісних характеристик робочого середовища. Це середовище у приміщеннях (офісах) в основному характеризується такими фізичними параметрами, як температура, вологість та електричний опір підлоги. Фізико-хімічні показники включають інформацію про

вміст у повітрі іонів та різноманітних забруднювачів, але так само деякі інші якісні характеристики середовища.

Висока температура повітря негативно позначається на функціональному стані людини. Хоч генерація теплоти дисплеєм досягає критичного рівня тільки у саму теплу пору року, треба створювати комфортні теплові умови постійно.

2.2 Розробка заходів із охорони праці

2.2.1 Мікроклімат

Оптимальні та допустимі мікрокліматичні параметри у приміщеннях повинні враховувати специфіку технологічного процесу при використанні комп'ютерів. Згідно із діючими у нашій країні нормативними документами (ДСанПіН 3.3.2-007-98 у холодні періоди року температура повітря, швидкість його руху та відносна вологість повітря повинні відповідно складати: 22-24^{0}С; 0,1 м/с; 40-60%. Температура повітря може коливатись у межах з 21 до 25^{0}С при збереженні інших параметрів мікроклімату.

В теплі періоди року температура повітря, його рухливість та відносна вологість повинні відповідно становити: 23-25^{0}С; 0,1-0,2 м/с; 40-60 %.

Оптимальним рівнем аероіонізації у зоні дихання користувача вважається вміст легких аерофонів обох знаків з 150 до 5000 у 1 см^3 повітря.

Нормалізуючий вплив на склад повітря робочої зони справляють примусова вентиляція, захисні екрани (оснащені заземленням) та застосування іонізаторів.

2.2.2 Виробничий шум

Деякі ВДТ є потенційними джерелами цілого ряду звуків, що містять як коливання, котрі можливо почути, так й коливання ультразвукового діапазону. Цей шум справляє негативний вплив на стан користувача, особливо при тривалому впливі. У користувача, діяльність якого пов'язана із переробкою інформації це виражається у знижені розумової працездатності, зростає число помилок, розвиток зорового втомлення, зміні відчуття кольорів, появі головного

болю, послаблення уваги. Нормованим параметром шуму на робочих місцях є рівень 50 дБ. Основними заходами боротьби із шумом є усунення чи ослаблення причин шуму в самому його джерелі у процесі проектування, використання інструментів звукопоглинання, раціональне планування виробничих приміщень.

2.2.3 Виробниче освітлення

Освітлення у приміщеннях із ВДТ містить існувати змішаним – природним та штучним. Природне освітлення повинно здійснюватись у виді бічного освітлення та відповідати нормам ДБН В.2.5-28-2006 «Природне й штучне освітлення».

При природному освітленні слід передбачити наявність сонцезахисних інструментів, що знижують перепади яскравостей поміж природним світлом та свіченням екрана ВДТ. ІЗ цією метою можливо використовувати плівки із металізованим покриттям чи жалюзі із вертикальними лам елями, що регулюються.

Штучне освітлення у приміщеннях із ВДТ треба здійснювати у виді комбінованої системи освітлення із використовуванням люмінесцентних джерел світла у світильниках загального освітлення. На робочих місцях містить існувати забезпечена рівномірна освітленість за поміччю переважно відбитого чи розсіяного світлорозподілу. Світлових відблисків із клавіатури, екрана та з інших частин ВДТ у напрямку очей користувача не повинно існувати.

Норма освітленості на робочих місцях складає 300-500лк.

2.2.4 Електробезпека

Причинами ураження працівника електрострумом можуть існувати:

- Випадковий дотик до струмоведучих частин, у результаті ведення робіт поблизу чи на цих частинах;
- Несправність захисних інструментів, котрими потерпілий доторкався до струмоведучих частин;

– Несподіване виникнення напруги крізь ушкодження ізоляції там, де в нормальних умовах його існувати не повинно;
– Контакт струмопровідного устаткування із проводом, що перебуває під напругою.

Значення сили струму, що проходить крізь організм людини, залежить з напруги, під котрою перебуває людина й з опору ділянки тіла, до якого прикладена ця напруга. Джерелом живлячої напруги є мережа змінного струму із напругою 229В, на яку поширюється ГОСТ 25861-83.

Задля попередження поразок електричним струмом треба чітко й у повному обсязі виконувати правила провадження робіт й правил технічної експлуатації. Треба виключити можливість доступу оператора до частин устаткування, що працює під небезпечною напругою, до неізольованим частинам, призначеним задля опрацювання при малій напрузі й не підключеним до захисного заземлення, але так само підводити електроживлення до ПЕОМ з розетки за поміччю спеціальної вилки із заземлюючим контактом.

2.2.5 Організація робочого місця

Обладнання й організація робочого місця із ВДТ містять забезпечувати відповідність конструкцій всіх елементів робочого місця та їхніх взаємного розташування, ергономічним вимогам, із урахуванням характеру й особливостей трудової діяльності (ДСанПіН 3.3.2-007-98).

Конструкція робочого місця й взаємне розташування всіх його елементів (сидіння, органи керування, засобу відображення інформації) відповідають антропометричним, фізіологічним й психологічним вимогам, але так само характеру опрацювання. Конструкція робочих меблів дає можливість забезпечувати можливість індивідуального регулювання їхніх відповідно до потреб працівника задля підтримки зручної пози. Робочий стіл повинен існувати пофарбований матовою фарбою. Дисплей розташований так, що його верхній край перебуває на рівні очей, на відстані близько 70 см, що укладається в припустимі рамки з 60 до 90 см. Частота мерехтіння екрана дорівнює 100 Гц, що

відповідає умові більше 70 Гц. Задля зниження нервово-емоційного напруження, стомлювання, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втомі доцільно впроваджувати виконання комплексу вправ, котрі наведені у Державних санітарних правилах й нормах опрацювання із візуальними терміналами електронно-обчислювальних машин ДСанПіН 3.3.2.007-98.

2.3 Пожежна безпека

Протипожежний захист приміщення забезпечується застосуванням автоматичної установки пожежної сигналізації, наявністю інструментів пожежогасіння, застосуванням основних будівельних конструкцій будинку із регламентованими межами вогнестійкості, організацією своєчасної евакуації людей.

До інструментів гасіння пожежі відносяться внутрішні пожежні водопроводи (крани – ПК), вогнегасники (вуглекислотні та порошкові), сухий пісок тощо.

ВИСНОВКИ

Дана випускна робота присвячена проблемі поліпшення цифрових зображень із низькою розділеною можливістю, рішення якої за поміччю створених програмних інструментів надає виконувати їхніх розтягування та інтерполяцію без пікселізації.

Розглянуті у роботі теоретичні основи розтягування зображень, векторизації, побудови сплайнів, здійснення дерева когерентності поміж сусідніми пікселами, побудови й спрощення діаграм Діріхле дозволили побудувати найбільш оптимальний метод.

Дослідження існуючих алгоритмів та аналогічних програмних інструментів задля рішення даної задачі дозволили визначити сильні та слабкі місця існуючих методів інтерполювання та розтягування зображень, адже жоден із розглянутих інструментів не давав бажаних результатів та мав обмеження кратності розтягування.

Реалізований метод призначений задля інтерполювання, розтягування та поліпшення якості зображень із низькою розділеною можливістю, але програмне забезпечення реалізоване у виді library, що дозволить із легкістю інтегрувати її у піксельні ігри чи використовувати задля адаптації застарілого програмного забезпечення до сучасних HD-дисплеїв. Програмний застосунок задля демонстрації можливостей розробленої library мовою програмування C++ надає відобразити окремі етапи опрацювання зображення з початкового до вихідного.

Застосування library, створеної за реалізованим алгоритмом поліпшення цифрових зображень, надає достатньо якісно масштабувати вхідне зображення низької роздільної здатності у довільну число раз із реалізацією градієнтних переходів довільної складності. Висока швидкість опрацювання створеної library досягається при мінімальному використанні системних ресурсів, що надає використати її й на мобільних пристроях із операційними системами Android та ін.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХДЖЕРЕЛ

1. Комп'ютерна графіка: конспект лекцій / Укладач: Скиба О.П. – Тернопіль: Тернопільський національний технічний університет імені Івана Пулюя, 2019. – 88с.
2. Ю.О. Гришко, О.С. Шкільняк. Алгоритми обчислювальної геометрії. Навчальний посібник. – 2020 – 153с.
3. Веселовська Г.В., Ходакова В.Є.: Комп'ютерна графіка. Навч. пос. – К.: Кондор, 2015. – 584с.
4. Галісєєв Г. Системне програмування. – К.: Вид-во Університет "Україна", 2019. – 113 с.
5. Зайцев В.Г., Дробязко І.П. Операційні системи: навч. Посібник для студ. – К.: КПІ ім. Ігоря Сікорського, 2019. – 240 с.
6. Мосіюк О.О., Федорчук А.Л. Операційні системи та системне програмування: навчально-методичний посібник. Житомир: Вид-во ЖДУ ім. Івана Франка, 2022. – 76 с.
7. Трофименко О.Г. С++. Алгоритмізація та програмування : підручник / О.Г. Трофименко, Ю.В. Прокоп, Н.І. Логінова, О.В. Задерейко. 2-ге вид. перероб. і доповн. – Одеса : Фенікс, 2019. – 477 с.
8. Chen J.-J. System Programming Vol I / Jin-Jwei Chen. – Publisher, 2020. – 727 p. – ISBN: 978-1736193006.
9. Chen J.-J. System Programming Vol II / Jin-Jwei Chen. – Publisher, 2020. – 745 p. – ISBN : 978-1736193013.
10. Stroustrup B. A Tour of C++ (Second Edition). – Addison-Wesley, 2018. – 240 p. – ISBN 978-0-13-499783-4.
11. Бібліотека MSDN. Електронний ресурс: http://msdn.microsoft.com/ru-ru/library/default.aspx
12. Електронний ресурс: https://www.researchgate.net/publication/220184192_Depixelizing_Pixel_Art

ДОДАТОК А

Фрагмент тексту програми для покращення зображень мовою C++

Клас Depixelizing та його методи

```
class Depixelizing { public:
Depixelizing();
~Depixelizing();
cv::Mat depixelize(const cv::Mat& , float); private:
int m_width; // image width int m_height; // image height
cv::Mat m_image; // input image float m_scale; // scale factor
std::vector<unsigned char> m_connections; // similarity graph
// (i, j) -> (i * m_w + j)
std::list<Vertex> m_vertexes; std::vector<std::list<std::list<Vertex>::iterator>> m_curvesVertexes; std::list<Edge> m_edges;
std::vector<std::vector<std::list<Vertex>::iterator>> m_pixelsToCells; std::vector<std::vector<Vertex>> m_curves;
//building
void buildSimilarityGraph();
void deleteFullyConnectedBlocks();
//visualization
void showSimilarityGraph(); void showCellGraph();
void showCellImage(); // show and save in cell_image.png void showSplines();
cv::Mat render();
//heuristics
void resolveHeuristics();
void searchNeighbours(int &x, int &y, const std::vector <std::vector <int>>& a); // for curves heuristic
void search(int y, int x, int sy, int sx, std::vector <std::vector <int>>& a); // for sparse pixels heuristic
int curvesHeuristic(int x, int y);
int sparsePixelsHeuristic(int x, int y); int islandsHeuristic(int x, int y); unsigned char valence(unsigned char node);
//building cells void buildCells();
void combineCellsDiagonals();
//reconstructing cells void remove2ValenceNodes();
void reconstructPixelsToCells();
//combining cells
bool connected(int ind1, int ind2); void combineCells();
//searching curves std::list<Vertex>::iterator searchCurve(
std::list<Vertex>::iterator it, std::list<Vertex>::iterator it2);
void findAllCurves();
//splines
void buildSplines(int steps);
cv::Mat buildZonesMap(float scale, int w, int h);
};
// depixelizing.cpp #include "depixelizing.h"
Depixelizing::Depixelizing() : m_scale(20) {
}
Depixelizing::~Depixelizing() {
}
cv::Mat Depixelizing::depixelize(const cv::Mat& image, float scaleFactor) {
//initialization m_width = image.cols; m_height = image.rows;
m_image = image.clone(); m_scale = scaleFactor;
//depixelizing buildSimilarityGraph(); showSimilarityGraph(); deleteFullyConnectedBlocks(); showSimilarityGraph();
resolveHeuristics(); showSimilarityGraph(); buildCells(); showCellGraph(); combineCellsDiagonals(); showCellGraph();
remove2ValenceNodes(); showCellGraph(); reconstructPixelsToCells(); showCellImage(); combineCells(); showCellGraph();
findAllCurves();
buildSplines((int) scaleFactor); showSplines();
return render();
}
// cells.h #pragma once
#include <list>
struct Edge;
struct Vertex {
// coordinates float x; float y;
//additional variable for searching curves
//true - if vertex is already visited bool visited;
std::list<std::list<Edge>::iterator> edges; void addVertex(float x, float y) {
this->x = x; this->y = y;
}
};
struct Edge {
//vertexes on the edge std::list<Vertex>::iterator v1; std::list<Vertex>::iterator v2;v2) {
//indexes of pixels-neighbours
// -1 - if there are no neighbour int ind1;
int ind2;
void addEdge(int ind1, int ind2, std::list<Vertex>::iterator v1, std::list<Vertex>::iterator
this->ind1 = ind1; this->ind2 = ind2; this->v1 = v1; this->v2 = v2;
}
};
// buildSimilarityGraph.cpp #include "depixelizing.h"
void Depixelizing::buildSimilarityGraph() { cv::Mat imgYUV(m_image.size(), CV_8UC3);
cv::cvtColor(m_image, imgYUV, cv::COLOR_BGR2YUV);
```

```
m_connections.resize(m_width * m_height);
//building
for (int i = 0; i < m_height; ++i) { for (int j = 0; j < m_width; ++j) {
cv::Vec3b& currentPixel = imgYUV.at<cv::Vec3b>(i, j);
//TOP
if (i > 0) {
cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i - 1, j);
if (abs(currentPixel[0] - neighbourPixel[0]) < 48 && abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
abs(currentPixel[2] - neighbourPixel[2]) < 6) {
m_connections[i * m_width + j] |= NEIGHBOUR_TOP;
}
}
//TOP_RIGHT
if (j < m_width - 1 && i > 0) {
cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i - 1, j + 1);
if (abs(currentPixel[0] - neighbourPixel[0]) < 48 && abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
abs(currentPixel[2] - neighbourPixel[2]) < 6) {
m_connections[i * m_width + j] |= NEIGHBOUR_TOP_RIGHT;
}
}
//RIGHT
if (j < m_width - 1) {
cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i, j + 1);
if (abs(currentPixel[0] - neighbourPixel[0]) < 48 && abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
abs(currentPixel[2] - neighbourPixel[2]) < 6) {
m_connections[i * m_width + j] |= NEIGHBOUR_RIGHT;
}
}
//BOTTOM_RIGHT
if (j < m_width - 1 && i < m_height - 1) {
cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i + 1, j + 1);
if (abs(currentPixel[0] - neighbourPixel[0]) < 48 && abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
abs(currentPixel[2] - neighbourPixel[2]) < 6) {
m_connections[i * m_width + j] |= NEIGHBOUR_BOTTOM_RIGHT;
}
}
//BOTTOM
if (i < m_height - 1) {
cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i + 1, j);
if (abs(currentPixel[0] - neighbourPixel[0]) < 48 && abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
abs(currentPixel[2] - neighbourPixel[2]) < 6) {
m_connections[i * m_width + j] |= NEIGHBOUR_BOTTOM;
}
}
//BOTTOM_LEFT
if (j > 0 && i < m_height - 1) {
cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i + 1, j - 1);
if (abs(currentPixel[0] - neighbourPixel[0]) < 48 && abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
abs(currentPixel[2] - neighbourPixel[2]) < 6) {
m_connections[i * m_width + j] |= NEIGHBOUR_BOTTOM_LEFT;
}
}
//LEFT
if (j > 0) {
cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i, j - 1);
if (abs(currentPixel[0] - neighbourPixel[0]) < 48 && abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
abs(currentPixel[2] - neighbourPixel[2]) < 6) {
m_connections[i * m_width + j] |= NEIGHBOUR_LEFT;
}
}
//TOP_LEFT
if (j > 0 && i > 0)
{
cv::Vec3b& neighbourPixel = imgYUV.at<cv::Vec3b>(i - 1, j - 1);
if (abs(currentPixel[0] - neighbourPixel[0]) < 48 && abs(currentPixel[1] - neighbourPixel[1]) < 7 &&
abs(currentPixel[2] - neighbourPixel[2]) < 6) {
m_connections[i * m_width + j] |= NEIGHBOUR_TOP_LEFT;
}
}
}
}
}
void Depixelizing::deleteFullyConnectedBlocks() { for (int i = 0; i < m_height - 1; ++i) {
for (int j = 0; j < m_width - 1; ++j) {
if ((m_connections[i * m_width + j] & NEIGHBOUR_RIGHT) && (m_connections[i * m_width + j] &
NEIGHBOUR_BOTTOM) && (m_connections[(i + 1) * m_width + j] & NEIGHBOUR_RIGHT) &&
(m_connections[i * m_width + (j + 1)] & NEIGHBOUR_BOTTOM)) {
m_connections[i * m_width + j] &= ~NEIGHBOUR_BOTTOM_RIGHT; m_connections[(i+1) * m_width + (j+1)]
&= ~NEIGHBOUR_TOP_LEFT; m_connections[(i + 1) * m_width + j] &= ~NEIGHBOUR_TOP_RIGHT;
m_connections[i * m_width + (j + 1)] &= ~NEIGHBOUR_BOTTOM_LEFT;
}
}
}
}
```

Модуль visualization.cpp

```
// visualization.cpp #include "depixelizing.h"
void Depixelizing::showSimilarityGraph() { int scale = int(m_scale);
cv::Mat graph(scale * m_height, scale * m_width, CV_8UC3, cv::Scalar(0,0,0)); for (int i = 0; i < m_height; ++i) {
for (int j = 0; j < m_width; ++j) {
if (m_connections[i * m_width + j] & NEIGHBOUR_BOTTOM) {
cv::line(graph, cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + i * scale),
cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + (i + 1) * scale), cv::Scalar(255,255,255));
}
if (m_connections[i * m_width + j] & NEIGHBOUR_BOTTOM_RIGHT) {
cv::line(graph, cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + i * scale),
cv::Point((scale + 1) / 2 + (j + 1) * scale, (scale + 1) / 2 + (i + 1) * scale), cv::Scalar(255,255,255));
}
if (m_connections[i * m_width + j] & NEIGHBOUR_BOTTOM_LEFT) {
cv::line(graph, cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + i * scale),
cv::Point((scale + 1) / 2 + (j - 1) * scale, (scale + 1) / 2 + (i + 1) * scale), cv::Scalar(255,255,255));
}
if (m_connections[i * m_width + j] & NEIGHBOUR_RIGHT) {
cv::line(graph, cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + i * scale),
cv::Point((scale + 1) / 2 + (j + 1) * scale, (scale + 1) / 2 + i * scale), cv::Scalar(255,255,255));
}
if (m_connections[i * m_width + j] & NEIGHBOUR_LEFT) {
cv::line(graph, cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + i * scale),
cv::Point((scale + 1) / 2 + (j - 1) * scale, (scale + 1) / 2 + i * scale), cv::Scalar(255,255,255));
}
if (m_connections[i * m_width + j] & NEIGHBOUR_TOP_RIGHT) {

cv::line(graph, cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + i * scale),
cv::Point((scale + 1) / 2 + (j + 1) * scale, (scale + 1) / 2 + (i - 1) * scale), cv::Scalar(255,255,255));
}
if (m_connections[i * m_width + j] & NEIGHBOUR_TOP) {
cv::line(graph, cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + i * scale),
cv::Point((scale + 1) / 2 + j * scale, (scale + 1) / 2 + (i - 1) * scale), cv::Scalar(255,255,255));
}
if (m_connections[i * m_width + j] & NEIGHBOUR_TOP_LEFT) {
cv::line(graph, cv::Point( (scale + 1) / 2 + j * scale, (scale + 1) / 2 + i * scale),
cv::Point((scale + 1) / 2 + (j - 1) * scale, (scale + 1) / 2 + (i - 1) * scale), cv::Scalar(255,255,255));
}
}
}
for (int i = 0; i < m_height; ++i) { for (int j = 0; j < m_width; ++j) {
cv::circle(graph, cv::Point(cvRound((scale + 1) / 2 + j * scale),
cvRound((scale + 1) / 2 + i * scale)), 1, cv::Scalar(0, 255, 255), 2);
}
}
cv::imshow("SimilarityGraph", graph); cv::waitKey(0);
}
void Depixelizing::showCellGraph() { const int scale = (int)m_scale;
cv::Mat img(m_height * scale + 10, m_width * scale + 10, CV_8UC3, cv::Scalar(0,0,0));
for (auto it = m_edges.begin(); it != m_edges.end(); ++it) { cv::line(img,
cv::Point(cvRound(it->v1->x * scale + 5), cvRound(it->v1->y * scale + 5)),
cv::Point(cvRound(it->v2->x * scale + 5), cvRound(it->v2->y * scale + 5)), cv::Scalar(255, 255, 255), 1);
cv::circle(img, cv::Point(cvRound(it->v1->x * scale + 5), cvRound(it->v1->y * scale + 5)), 1, cv::Scalar(200, 255, 0), 2);
cv::circle(img, cv::Point(cvRound(it->v2->x * scale + 5), cvRound(it->v2->y * scale + 5)), 1, cv::Scalar(200, 255, 0), 2);
}
cv::imshow("CellGraph", img); cv::waitKey(0);
}
void Depixelizing::showCellImage() { const float scale = m_scale;
cv::Mat cellImage(m_height * scale, m_width * scale, CV_8UC3, cv::Scalar(0,0,0));
for (int y = 0; y < m_height; ++y) { for (int x = 0; x < m_width; ++x) {
size_t ind = y * m_width + x; // index of current cell std::vector <cv::Point> points;
for (size_t i = 0; i < m_pixelsToCells[ind].size(); ++i)
{ points.push_back(cv::Point(static_cast<int>(m_pixelsToCells[ind][i]->x * scale),
static_cast<int>(m_pixelsToCells[ind][i]->y * scale)));
}
int vertexCount;
vertexCount = m_pixelsToCells[ind].size();
cv::Scalar c(m_image.at<cv::Vec3b>(y, x)[0], m_image.at<cv::Vec3b>(y, x)[1], m_image.at<cv::Vec3b>(y, x)[2]);
cv::fillPoly(cellImage, (const cv::Point**)&points, &vertexCount, 1, c); points.clear();
}
}
cv::imshow("cell_image", cellImage); cv::imwrite("cell_image.png", cellImage); cv::waitKey();
}
void Depixelizing::showSplines() { const int scale = (int)m_scale;
cv::Mat img(m_height * scale, m_width * scale, CV_8UC3, cv::Scalar(0,0,0));
for (size_t i = 0; i < m_curves.size(); ++i) {
for (size_t j = 0; j < m_curves[i].size() - 1; ++j) { cv::line(img,scale)),* scale)),
}
}
cv::Point(cvRound(m_curves[i][j].x * scale), cvRound(m_curves[i][j].y *
cv::Point(cvRound(m_curves[i][j + 1].x * scale), cvRound(m_curves[i][j + 1].y
cv::Scalar(255, 255, 255), 1);
cv::imshow("Splines", img); cv::waitKey();
```

```
}
cv::Mat Depixelizing::render() { float scale = m_scale;
int w = static_cast<int>(m_width * scale); int h = static_cast<int>(m_height * scale);
cv::Mat map = buildZonesMap(scale, w, h);
cv::Mat res(h, w, CV_8UC3, cv::Scalar(0, 0, 0)); cv::Mat out;
int shift = ceil(m_scale / 10); for (int y = 0; y < h; ++y) {
int sy = static_cast<int>(y / scale); int miny = sy - shift;
if (miny < 0) miny = 0;
int maxy = sy + shift;
if (maxy >= m_height) maxy = m_height - 1; for (int x = 0; x < w; ++x) {
int sx = static_cast<int>(x / scale); int minx = sx - shift;
if (minx < 0) minx = 0; int maxx = sx + shift;
if (maxx >= m_width) maxx = m_width - 1;
unsigned short currId = map.at<unsigned short>(y, x); cv::Vec3f c(0.f, 0.f, 0.f);
int count = 0;
for (int yy = miny; yy <= maxy; ++yy) {
int syy = static_cast<int>(yy * scale + scale / 2); if (syy >= h)
continue;
for (int xx = minx; xx <= maxx; ++xx) {
int sxx = static_cast<int>(xx * scale + scale / 2); if (sxx >= w)
continue;
if (map.at<unsigned short>(syy, sxx) == currId) { cv::Vec3b cc = m_image.at<cv::Vec3b>(yy, xx);
++count;
c[0] += cc[0];
c[1] += cc[1];
c[2] += cc[2];
}
}
}
c *= 1.f / static_cast<float>(count);
res.at<cv::Vec3b>(y,x)=cv::Vec3b(static_cast<uchar>(c[0]), static_cast<uchar>(c[1]), static_cast<uchar>(c[2]));
}
}
// Blur
int ks = shift;
int blurIterations = 3;
for (int iter = 0; iter < blurIterations; ++iter) { for (int y = 0; y < h; ++y) {
int miny = y - ks;
if (miny < 0) miny = 0; int maxy = y + ks;
if (maxy >= h) maxy = h - 1;
for (int x = 0; x < w; ++x) { int minx = x - ks;
if (minx < 0) minx = 0; int maxx = x + ks;
if (maxx >= w) maxx = w - 1; int count = 0;
cv::Vec3f c(0.f, 0.f, 0.f);
unsigned short currId = map.at<unsigned short>(y, x); for (int yy = miny; yy <= maxy; ++yy) {
for (int xx = minx; xx <= maxx; ++xx) {
if (map.at<unsigned short>(yy, xx) == currId) { cv::Vec3b cc = res.at<cv::Vec3b>(yy, xx); c[0] += static_cast<float>(cc[0]);
c[1] += static_cast<float>(cc[1]); c[2] += static_cast<float>(cc[2]);
++count;
}
}
}
c *= 1.f / static_cast<float>(count);
res.at<cv::Vec3b>(y, x) = cv::Vec3b(static_cast<uchar>(c[0]), static_cast<uchar>(c[1]), static_cast<uchar>(c[2]));
}
}
}
return res;
}
```

Модуль demo.cpp для тестування створеного алгоритма

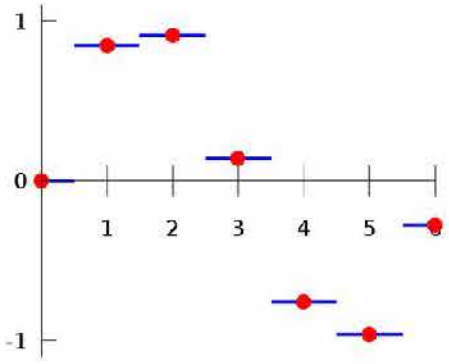
```
// demo.cpp
#include "depixelizing.h" #include <opencv2/opencv.hpp> #include <iostream>
#include <string>
int main() {
std::string fileName;
std::cout << "Enter pixel art file name: "; std::cin >> fileName;
cv::Mat img = cv::imread(fileName); cv::Mat input;
if (img.empty()) {
std::cout << "ERROR: failed to load image" << std::endl; return -1;
}
std::cout << "Enter scale factor: "; float scale = 20;
std::cin >> scale; if (scale < 2.f) {
scale = 2.f;
}
cv::resize(img, input, input.size(), scale, scale, cv::INTER_NEAREST); cv::imshow("Input image X scale factor", input);
cv::waitKey(0); Depixelizing pix;
cv::Mat res = pix.depixelize(img, scale); cv::imshow("Output_image", res); cv::imwrite("output.png", res); cv::waitKey(0);
return 0;
}
```

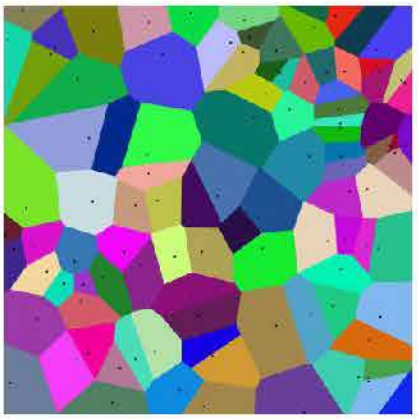
ДОДАТОК Б

Слайди мультимедійної презентації

Масштабування растрового цифрового зображення

Зображення у стилі Pixel Art



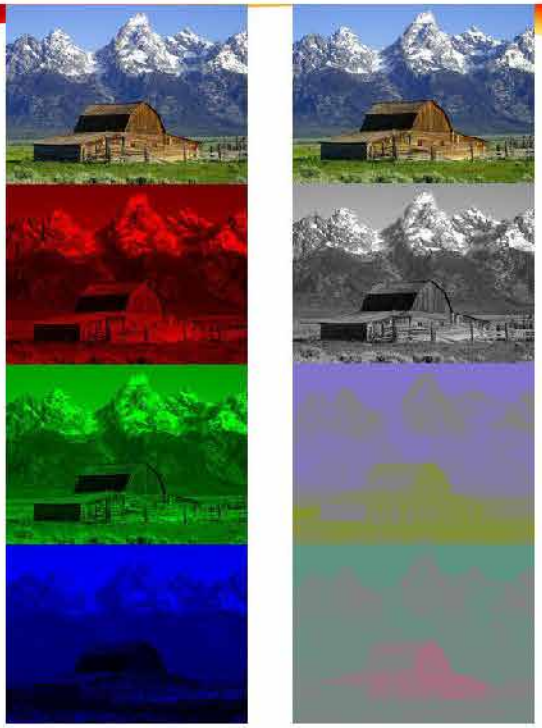


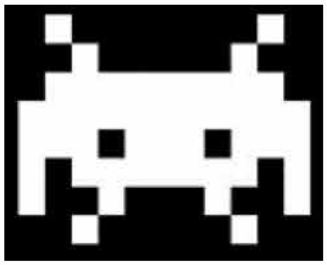
Інтерполяція за алгоритмом Найближчого сусіда

Етапи обробки зображення:

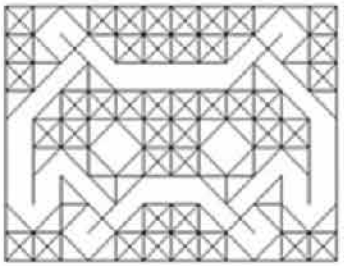
- етап створення графу зв'язності між сусідніми пікселями;
- етап створення і спрощення діаграми Вороного;
- етап створення сплайнів;
- етап розфарбування вихідного зображення.

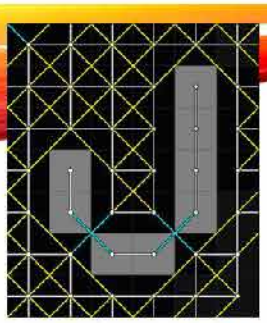
Розділення каналів зображення
у кольорових форматах
RGB (зліва) та YUV (справа)

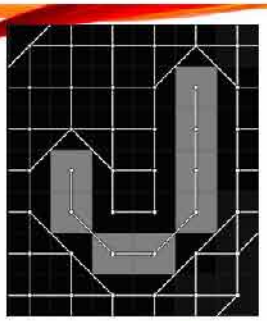




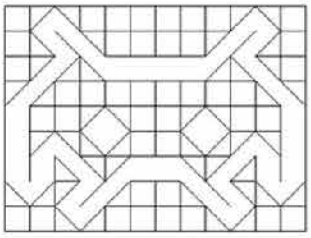
Побудова графу
зв'язності пікселів
з діагональними
перетинами



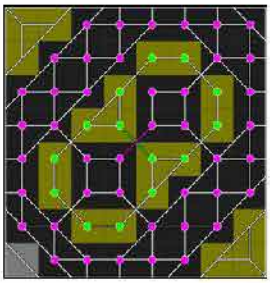


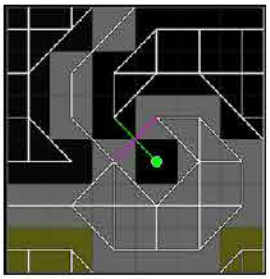


Евристика ламаної лінії з подальшим видаленням діагоналей

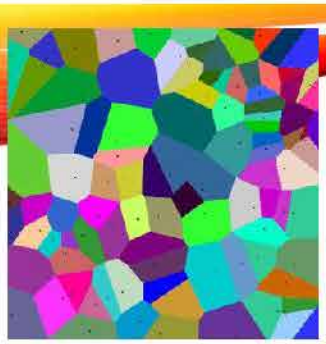


Граф зв'язності пікселів без діагональних перетинів

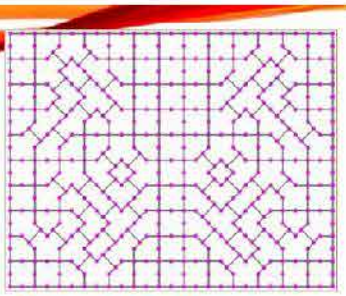




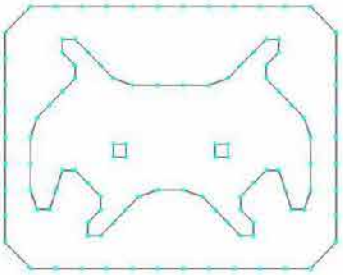
Евристики при визначенні плану і фону та одновалентного вузлу



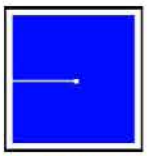
Діаграми Вороного випадкової множини точок на плоскості



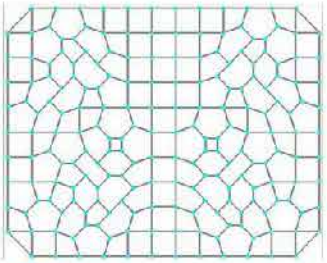
Побудова діаграми Вороного з графу



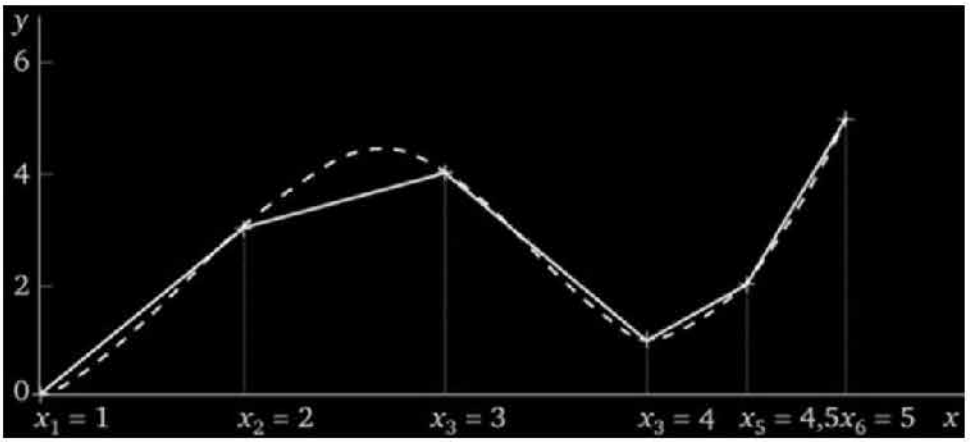
Контур зображення з ламаних ліній



Формування діаграми Вороного навколо області без діаголалі та з діагоналлю

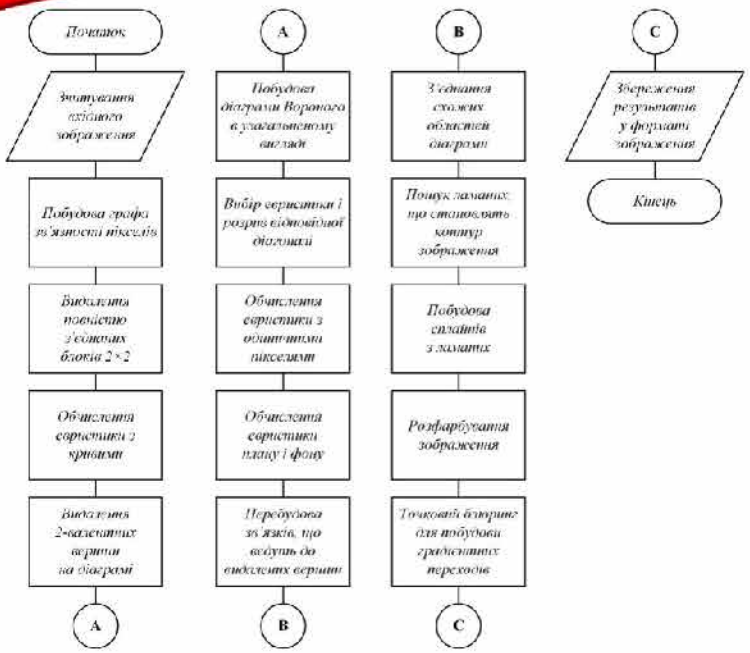


Побудова спрощеної діаграми Вороного

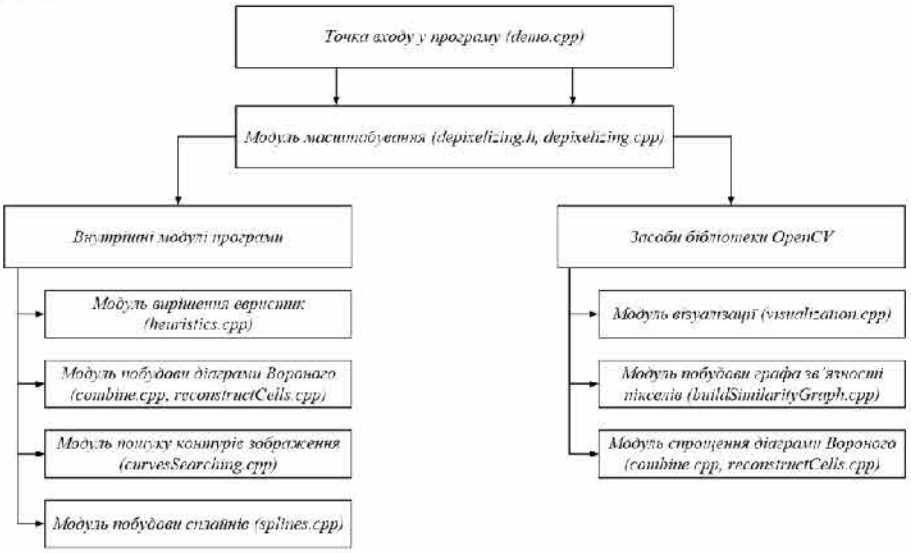


Створення кубічного сплайну для ламаної лінії

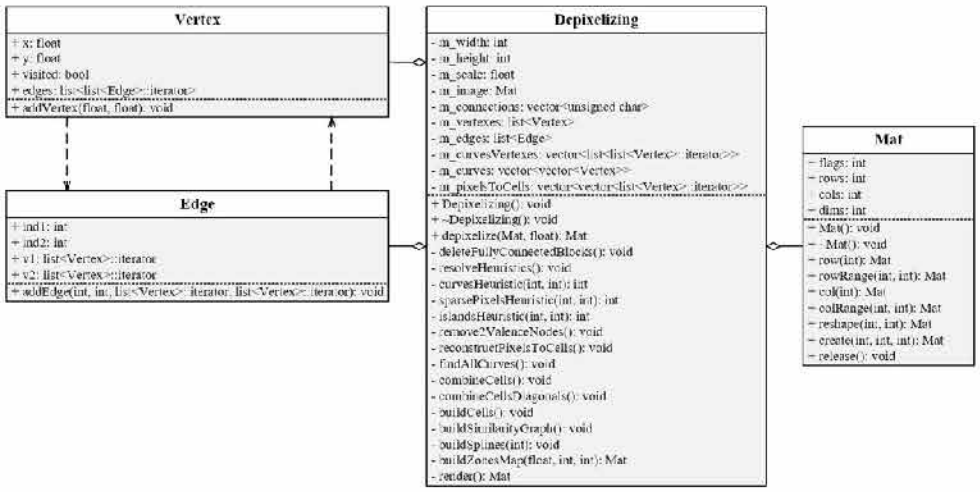
Узагальнена блок-схема алгоритму обробки зображення

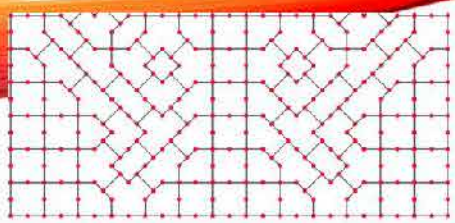


Діаграма взаємодії модулів програми для обробки піксельних зображень

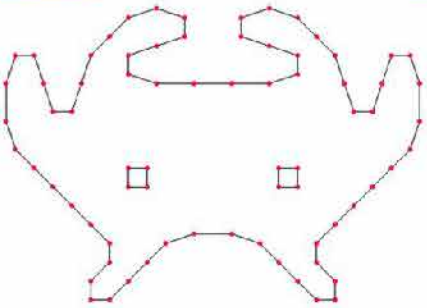


Діаграма класів для реалізованої програми

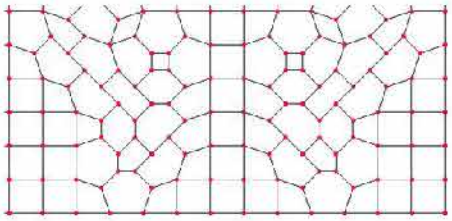




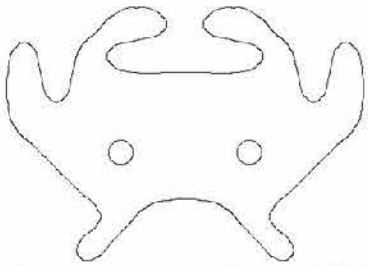
Додавання нових вершин та зсув існуючих у діаграмі Вороного



Ламані, що складають контур зображення



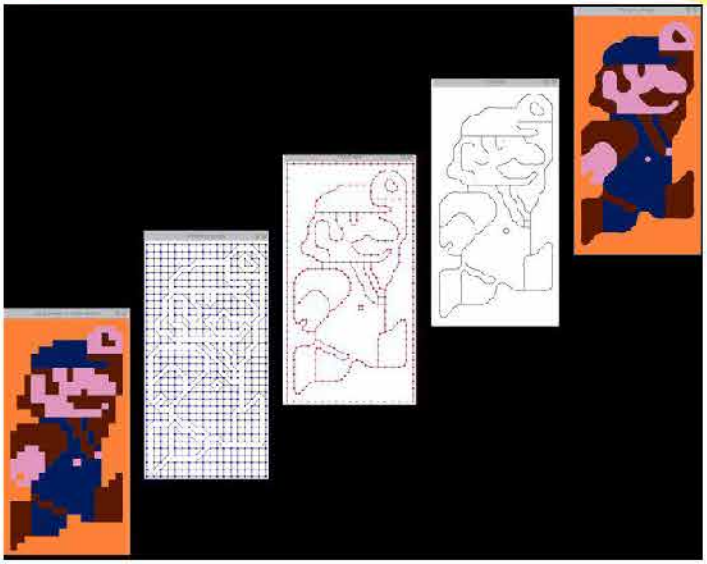
Отримання діаграми Вороного спрощеного типу



Набір кривих, що описують контур зображення

```
/PixelArt$ cmake --build cmake-build-release --target PixelArt
Scanning dependencies of target PixelArt
[  9%] Building CXX object CMakeFiles/PixelArt.dir/demo.cpp.o
[ 18%] Linking CXX executable PixelArt
[100%] Built target PixelArt
/PixelArt$ cd cmake-build-release/
/PixelArt/cmake-build-release$ ./PixelArt
Enter pixel art file name: inputs/12.png
Enter scale factor: 28
```

Інтерфейс додатку для демонстрації роботи реалізованого алгоритму покращення цифрових зображень



Покрокове відображення результатів роботи алгоритму покращення зображень

Результати порівняння зображень:

а) початкове *б) у Adobe Live Trace* *в) за створеним алгоритмом*

Результати порівняння зображень:

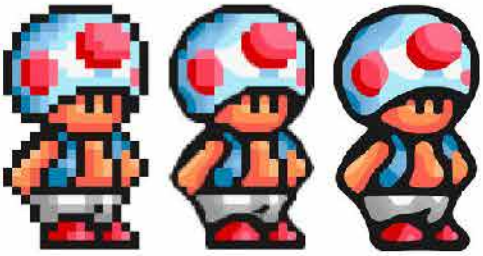
а) початкове *б) бібліотекою hq4x* *в) за створеним алгоритмом*

Результати порівняння зображень:

а) початкове *б) бібліотекою hq4x* *в) за створеним алгоритмом*

Результати порівняння зображень:

а) початкове *б) у Photo Zoom 4* *в) за створеним алгоритмом*



Результати порівняння зображень:

а) початкове *б) бібліотекою hq4x* *в) за створеним алгоритмом*

Результати порівняння зображень:

а) початкове *б) бібліотекою EPX* *в) за створеним алгоритмом*

ВСП «Одеський технічний фаховий коледж ОНТУ»

ВІДГУК

керівника про кваліфікаційну роботу бакалавра

Самолиги Дмитра Максимовича

(прізвище, ім'я та по батькові)

Спеціальність *123 "Комп'ютерна інженерія"*

Тема кваліфікаційної роботи *Аналіз ефективності методів покращення цифрових зображень*

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) Обсяг і якість виконання роботи (графічного матеріалу і розрахунково-пояснювальної записки) *Випускна робота виконана відповідно технічному завданню. Пояснювальна записка до випускної роботи містить 77 сторінок. У пояснювальній записці розглянуто проблему підвищення якості цифрових зображень, описані методи покращення якості цифрових зображень з низькою роздільною здатністю та реалізовані алгоритми, які дозволяють виконувати масштабування і інтерполяцію таких зображень. Графічна частина складається з 17 слайдів, оформлених у вигляді презентації, передбачених технічним завданням. Якість виконання пояснювальної записки та слайдів добра, розробку виконано у повному обсязі.*

б) Самостійність роботи

Протягом виконання випускної бакалаврської роботи Самолига Дмитро поступово та послідовно виконував всі етапи, проявив ініціативу у створенні загальної концепції та реалізації випускної роботи. Всі роботи він виконував самостійно, з оглядом на рекомендації керівника.

в) Теоретична підготовка здобувача освіти

Самолига Дмитро під час роботи над випускною бакалаврською роботою вивчив і опрацював достатню кількість літературних джерел за даною тематикою.

Вважаю, що теоретична підготовка здобувача освіти достатня і він готовий до захисту роботи.

г) Вміння розв`язувати виробничі і конструкторські питання на базі останніх досліджень науки і техніки, передових методів виробництва

Під час виконання роботи Самолига Дмитро мав змогу самостійно приймати окремі рішення з програмної реалізації алгоритму та показав вміння організовано працювати над поставленою задачею, використовувати сучасні системи програмування та графічні інтерфейси, працювати у спеціалізованих графічних пакетах та САПР

Оцінка розрахункової частини *Добре*

Оцінка графічної частини *Відмінно*

Загальна оцінка *Відмінно*

Прізвище, ім`я, по батькові *Кірєєв Ігор Анатолійович*

Місце роботи і посада керівника роботи

Державний університет інтелектуальних технологій і зв'язку, доцент каф. інформаційної безпеки та передачі даних

Підпис

«12» червня 2024р.

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра здобувача освіти

відділення комп'ютерних систем

Самолиги Дмитра Максимовича

(прізвище, ім'я та по батькові)

Спеціальність 123 "Комп'ютерна інженерія"

Освітня програма «Комп'ютерна інженерія»

Керівник дипломного проекту (роботи) Кірєєв Ігор Анатолійович

(прізвище, ім'я та по батькові)

Тема дипломного проекту (роботи) Аналіз ефективності методів покращення цифрових зображень

Обсяг розрахунково-пояснювальної записки 77 сторінок

Обсяг графічної (презентаційної) частини 17 аркушів (слайдів)

ХАРАКТЕРИСТИКА ДИПЛОМНОГО ПРОЕКТУ (РОБОТИ)

а) заключення про ступінь відповідності виконаного дипломного проекту (роботи) завданню

Представлена на рецензію кваліфікаційна робота бакалавра повністю відповідає меті проектування та технічному завданню. Тематика кваліфікаційної роботи є актуальною для своєї галузі та присвячена проблемі підвищення якості цифрових зображень, методам покращення якості цифрових зображень з низькою роздільною здатністю.

б) характеристика виконання кожного розділу дипломного проекту (роботи)

Кваліфікаційна робота складається зі вступу, двох розділів, висновків, переліку використаних джерел. У основному розділі розглянуті питання підвищення якості цифрових зображень, описані методи покращення якості цифрових зображень з низькою роздільною здатністю та реалізовані алгоритми, які дозволяють виконувати масштабування і інтерполяцію таких зображень

в) оцінка якості виконання пояснювальної записки та графічної частини дипломного проекту (роботи) Графічна частина виконана на достатньо високому рівні у вигляді презентації із використанням офісного пакету Microsoft PowerPoint та Visio. Пояснювальна записка виконана охайно та у відповідності до норм оформлення документів із використанням офісного пакету Microsoft Word. Загальна якість виконання документації – добра, академічного плагіату у роботі не виявлено

г) перелік позитивних якостей дипломного проекту (роботи)

1. Детально описано мету та цілі аналізу;
2. Проведено ретельний аналіз проблеми підвищення якості цифрових зображень з низькою роздільною здатністю
3. Реалізовано алгоритми, які дозволяють виконувати масштабування і інтерполяцію таких зображень, що представляє практичний інтерес.

д) основні недоліки дипломного проекту (роботи)

1. При тестуванні реалізованого методу покращення зображень варто було б також використовувати зображення з градієнтними переходами.
2. Варто було б передбачити візуальний інтерфейс для застосунку.

Оцінка розрахункової частини Добре

Оцінка графічної частини Відмінно

Загальна оцінка Відмінно

Прізвище, ім'я, по батькові рецензента к.т.н. Селіванова Алла Віталіївна

Місце роботи і посада рецензента Одеський національний технологічний університет, декан факультету комп'ютерної інженерії, програмування та кіберзахисту

Підпис:

« 14 » 06 2024 р.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григорівна Краснокутська

ID перевірки:
1016271367

Дата перевірки:
21.05.2024 22:38:51 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
21.05.2024 23:50:35 EEST

ID користувача:
100011688

Назва документа: **2БКС-28_Дмитро_Самолига**

Кількість сторінок: **58** Кількість слів: **9365** Кількість символів: **68536** Розмір файлу: **3.26 MB** ID файлу: **1016060753**

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

14.7%
Схожість

Найбільша схожість: **7.33%** з Інтернет-джерелом (https://ela.kpi.ua/bitstream/123456789/34626/1/Myroniuk_bakalavr.pd.

14.7% Джерела з Інтернету | 585 ..... Сторінка 60

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Підозріле форматування | 12 сторінок

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Самолига Дмитро Максимович,
здобувач освіти гр. 2БКС-28, та

Кірєєв Ігор Анатолійович,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Аналіз ефективності методів покращення цифрових зображень» (автор роботи – Самолига Д.С., керівник роботи – Кірєєв І.А.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Самолига Д.С. /

Керівник _____ / Кірєєв І.А. /

«13» червня 2024 р.