МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Магістерської та аспірантської підготовки

Кафедра інформаційних технологій

КОМПЛЕКСНА МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Аналіз та програмна реалізація алгоритмів пошуку та редагування зображень з урахуванням контенту»

Склад:

Частина 1. «Розробка підсистеми модифікації зображень за їх контентом»

Виконавець: Іщук А.О.
(П.І.Б.)

Керівник: Коваленко Л.Б.
(П.І.Б.)

Частина 2. «Розробка підсистеми систематизації зображень за їх контентом»

Виконавець: Шестопалов Д.С.
(П.І.Б.)

Керівник: Коваленко Л.Б.
(П.І.Б.)

Староста роботи: Іщук А.О.
(П.І.Б.)

Провідний керівник проекту: к.геогр.н., доц. Коваленко Л.Б.
(П.І.Б.)

Рецензент: к.т.н., доц. Гнатовська Г.А.
(П.І.Б.)

ОДЕСА 2018

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Магістерської та аспірантської підготовки

Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка підсистеми модифікації зображень за їх контентом»

Виконала студентка 2 року групи МК- 61
спеціальності 122 Комп'ютерні науки

Іщук Аліна Олександрівна

Керівник к.геог.н., доц.
Коваленко Людмила Борисівна

Консультант

Рецензент к.т.н., доц.
Гнатовська Ганна Арнольдівна

Одеса 2018

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Скорочення

CMYK – Cyan, Magenta, Yellow, BlacK color – субтрактивна колірна модель

CIE – International Commission on Illumination – міжнародна комісія по освітленню

EPX – Eric's Pixel eXpansion – піксельне збільшення

HoG – Histogram of Oriented Gradients – гістограма спрямованих градієнтів

IBM – International Business Machines corporation

RGB – Red, Green, Blue – адитивна колірна модель

PC – Personal computer – персональний комп'ютер

PHP – PHP: Hypertext Preprocessor – PHP: гіпертекстовий препроцесор

ROI – Regions Of Interest

WIC – Wavelet Image Compression

WPF – Windows Presentation Foundation – графічна (презентаційна) підсистема у складі .NET Framework для побудови клієнтських застосувань Windows

Терміни

Aliasing – «сходовий ефект», поява нерівномірності на різких діагональних кордонах зображення.

Ringing або Overshooting – виникнення хвильових перешкод близько різких меж на зображенні.

Seam – оптимальний з'єднаний шлях пікселів на одному зображенні зверху вниз або зліва направо, де оптимальність визначається енергетичною функцією зображення, яка розраховує важливість пікселів.

Subpixel shift – субпіксельне зрушення зображення, викликане особливостями реалізації інтерполяційних алгоритмів.

Unsharpening – розмивання або втрата чіткості зображення після масштабування.

YCbCr – сімейство колірних просторів, які використовуються для передачі кольорових зображень в компонентному відео і цифрової фотографії

ВСТУП

Багато галузей техніки, що мають відношення до отримання, обробки, зберігання та передачі інформації, в значній мірі орієнтуються в даний час на розвиток систем, в яких інформація має характер зображень. Рішення наукових і інженерних задач при роботі з візуальними даними вимагає особливих зусиль, що спираються на знання специфічних методів. Зміна розміру зображення є стандартним інструментом у багатьох додатках для обробки зображень. Вибір методу – це первинне завдання, яке повинен вирішити розробник програмних засобів стиснення даних. Вибір залежить від типу даних, які потрібно буде обробляти, апаратних ресурсів, вимог до ступеня стиснення і обмежень на час роботи програми.

Прості методи включають масштабування та обрізку, які призводять до втрати важливих особливостей зображення або спотворень. Ефективніша зміна розміру зображення може бути досягнута лише беручи до уваги контент зображення, а не лише геометричні обмеження. Одним з таких підходів є Seam carving, що дозволяє змінювати розмір зображення за рахунок пошуку так званих сімів найменш важливих пікселів та подальшого їх видалення чи дублювання. Тобто Seam carving дозволяє видалити пікселі з нецікавих частин зображення, зберігаючи при цьому важливий вміст. Seam carving може змінювати розмір зображення, вирізаючи чи вставляючи пікселі в різні частини зображення. Цей алгоритм використовує функцію енергії, яка визначає, наскільки важливі пікселі. Окрім зміни розміру цей підхід застосовується і для видалення зайвих об’єктів, визначених користувачем. Зміна розміру зображень може бути обумовлена різними факторами, такими як перегляд на різних пристроях з різними роздільними здатностями, розміщення у інтернеті, використання у дизайнерських рішеннях тощо.

Метою даної магістерської роботи є аналіз та програмна реалізація алгоритмумодифікації зображень з урахуванням їх змістовного наповнення.

Для досягнення поставленої мети в роботі необхідно вирішити наступні завдання:

− виконати аналіз існуючих методів модифікації та масштабування зображень;

− обґрунтувати вибір методів, що будуть використовуватися для вирішення проблеми модифікації зображень;

− надати методологічні основи вирішення поставленої проблеми;

− виконати проектування програмного забезпечення;

− розробити алгоритми та програмні реалізації етапів процесу модифікації зображень;

− виконати тестування розробленого програмного забезпечення;

− провести дослідження якості отриманих результатів на різних наборах даних.

Структура магістерської роботи складається з вступу, трьох розділів, висновків, переліку посилань на 17 найменувань, додатків. Повний обсяг роботи становить 73 сторінки, містить 32 рисунка і 26 формул.

1 АНАЛІЗ АЛГОРИТМІВ МАСШТАБУВАННЯ ЗОБРАЖЕНЬ

1.1 Подання цифрових зображень

Перш ніж розглянути алгоритми масштабування зображень, необхідно визначити, що в подальшому буде розумітися під зображенням. Комп'ютерне зображення в його цифровому поданні є набором значень інтенсивностей світлового потоку, розподіленого по кінцевій площі.

Для простоти розглянемо спочатку монохромні зображення. Інтенсивність випромінюваної світлової енергії з одиниці поверхні в точці з координатами (ξ, η) зображення можна представити деяким числом $B(\xi, \eta)$. Одиничний елемент зображення, що характеризується певним значенням (ξ, η), називається пікселем, а величина $z = f(\xi, \eta)$ – яскравістю [1].

З математичної точки зору, зображення в градаціях сірого можна уявити як дійсну функцію I двох дійсних змінних x і y. Функція $I(x, y)$ зображення в загальному випадку визначається в прямокутної області, але для зручності досліджень в роботі все зображення визначаються в квадратних областях, тобто $x \in [0; W]$, а $y \in [0; H]$, де W – ширина зображення, а H – висота зображення і $W=H$.

Всі зображення можна поділити на дві групи: з палітрою і без неї. У зображень з палітрою в пікселі (одному зі звітів зображення – значення функції $I(x, y)$ для конкретного x_i і y_i) зберігається число – індекс в деякому одновимірному векторі кольорів, званому палітрою. Палітри зазвичай бувають 8, 16 і 256 – кольорів [1].

Зображення без палітри зазвичай бувають в певній системі подання кольору або в градаціях сірого. В градаціях сірого значення кожного з пікселів визначається як яскравість точки. Найбільш часто зустрічаються зображення з 2-ма, 16-ма і 256-ю рівнями сірого.

Якщо зображення представлено в якійсь колірної моделі, то кожен її піксель є структурою, що описує компоненти кольору. Найбільш поширеною системою, що використовується в електронних і комп'ютерних системах, є система RGB. У цій системі колір визначається як комбінація червоного, зеленого і синього кольору. І на кожну зі складових припадатиме по одному байту. Існують і інші колірні моделі, такі як CMYK, CIE, YUU і YCbCr [1,2].

RGB – це адитивна кольорова модель, тобто заснована на складання кольорів безпосередньо випромінюючих об'єктів. Метод адитивного змішування заснований на особливостях будови зорового аналізатора

людини, зокрема на такому явищі як метамерія. Сітківка людського ока містить три типи колбочок, які сприймають світло в фіолетово-синьою, зелено-жовтої і жовто-червоній частинах спектра. Таким чином, змішуючи в певному співвідношенні три основні кольори – червоний (red), зелений (green) і синій (blue), можна відтворити більшість сприймаються людиною кольорів.

Як правило, кожна колірна компонента позначається цифрами від 0 до 255. При цьому 0 для кожного кольору дає чорний колір (0,0,0) і 255 для кожного кольору – це білий колір (255, 255, 255). Всі інші кольори передаються варіацією цих трьох кольорів. При цьому переданих кольорів буде більше 16 мільйонів [1].

Для того щоб коректніше оцінювати ступінь стиснення зображення при застосуванні того чи іншого алгоритму стиснення до даного зображення вводиться поняття класу зображення [1].

Під класом цифрового зображення розуміється сукупність зображень, при застосуванні до яких, алгоритм стиснення дає якісно однаковий результат. Наприклад, для одного класу алгоритм стиснення дає добрий коефіцієнт стиснення, а для іншого класу зображень навпаки, збільшує обсяг стискання файлу [1].

Умовно можна виділити наступні класи зображень:

− зображення з невеликою кількістю кольорів і великими областями, заповненими одним кольором. У зображенні відсутні плавні переходи кольорів. До таких класів зазвичай відноситься ділова графіка, науково-технічна, інженерна або плакатна графіка;

− зображення з плавними переходами кольорів, побудовані на комп'ютері: графіка презентацій і віртуальні моделі;

− фотореалістичні зображення, отримані після цифрового фотографування, сканування, а також постобработки цих зображень.

Можна виділити і специфічні класи зображень, такі як рентгенівські знімки, томаграфічне зображення, радіолокаційні плани місцевості і т.п. Але для порівняння алгоритмів стиснення зображень завжди необхідно визначати клас зображень, з якими вони працюють.

У процесі роботи з зображеннями додатки, які здійснюють обробку, пред'являють різні вимоги до алгоритмів стиснення зображень. Через специфіку додатків такі вимоги іноді можуть суперечити один одному. У загальному випадку можна виділити наступні вимоги до алгоритмів стиснення зображень:

− високий ступінь компресії;

− висока якість стисненого зображення (дана вимога суперечить виконанню попередньої вимоги, тому завжди доводиться шукати компроміс між ступенем стиснення і якістю відновленого зображення);

− висока швидкість компресії (дана вимога актуально для додатків, що займаються кодуванням зображень в реальному масштабі часу: цифрових фотоапаратів, відеокамер);

− висока швидкість декомпресії (дана вимога актуально майже для всіх додатків).

− можливість показати приблизне зображення, не чекаючи повного його завантаження (дана вимога актуально для мережевих додатків і для додатків, що займаються передачею великих зображень).

− облік специфіки зображення (дана вимога реалізують алгоритми стиснення, засновані на визначенні «області особливого призначення» (ROI – regions of interest)).

1.2 Класифікація методів масштабування растрової графіки

Під масштабуванням зображення слід розуміти збільшення або зменшення розміру зображення зі збереженням його пропорцій. Цей прийом часто використовується в комп'ютерній графіці, обробці відеофайлів, реалізується в відеотехніки. Масштабування може здійснюватися кількома методами в залежності від типу графіки. Якщо робота ведеться з векторною графікою, то проблем з масштабуванням не відбувається. Однак, при роботі з растровим зображенням потрібно використовувати спеціальні алгоритми, які і будуть розглянуті далі. Растрова графіка погано масштабируема, при ресемплінгі зображення рисунок доводиться відрисовувати заново.

Масштабування растрових зображень є однією з найважливіших завдань поряд з виконаннями поворотів, зміною контрастності і т.д. При цьому актуальна втрата якості зображення. Зокрема, можливі суттєві викривлення геометрії дрібних деталей і поява помилкових візерунків, виникають деякі негативні ефекти.

1) Аліасинг (Aliasing) – ефект «ступінчастості» ліній. Пов'язаний з проблемами відображення ліній, які не паралельні одній з осей координат. Ступінчастість виникає, коли точки на лініях перетинають рядки або стовпці пікселів під невеликим кутом. Частина лінії шириною в один піксель може потрапити на один піксель екрану, а частина – на інший. Виникає

невизначеність: можна малювати цю частину як один піксель на одному ряду, один піксель на іншому ряду або зафарбовувати обидва пікселя. На жаль, всі три способи вносять добре помітні дефекти в зображення [2].

2) Розмиті зображення (Unsharpening). Даний ефект особливо небажаний там, де мають місце об'єкти з вираженими межами. Він пов'язаний з тим, що при збільшенні зображень ряд методів призводять до того, що кордон стає розмазаний, не чітким. Надалі це може не тільки спотворити зображення візуально, а й призведе до ускладнення процедури сегментації і розпізнавання [3].

3) Ефект Гіббса. На зображеннях виявляються ореоли біля різких перепадів інтенсивності. Незначне спотворення зображень людина може і не помітити, тоді як при великих втратах інформації, наприклад, при сильній зміні масштабу, даний артефакт може стати дратівливим. За негативного ефекту він співмірний з втратою кольоровості [4].

Існують різні методи масштабування зображень, які характеризуються різним співвідношенням швидкодії і якості. Проведемо порівняння деяких популярних методів масштабування зображень.

Найбільш популярними методами збільшення масштабу зображення є методи, засновані на інтерполяції кольорів пікселів. Принцип дії полягає в тому, що для кожної точки кінцевого зображення береться фіксований набір точок вихідного і інтерполюється відповідно до його взаємного становища і обраного фільтра. Кількість точок теж залежить від фільтра.

Загальноприйняті алгоритми інтерполяції прийнято ділити на дві категорії: адаптивні та неадаптивні. Адаптивні методи змінюються в залежності від предмета і завдання інтерполяції (різкі межі, гладка текстура і т.д.), тоді як неадаптивні методи обробляють всі пікселі однаково.

1.2.1 Неадаптивні методи інтерполяції растрової графіки

Неадаптивні алгоритми включають: метод найближчого сусіда, білінійної інтерполяції, бікубічної інтерполяції, сплайни, функцію кардинального синуса та інші. Залежно від складності, вони використовують від 0 до 256 (або більше) суміжних пікселів для інтерполяції. Природно, що чем більше використовується інтерполяційне вікно, тим більш точним стає перетворення цифрового зображення, але це досягається за рахунок значного збільшення часу обробки.

1.2.1.1 Метод найближчого сусіда

Метод найближчого сусіда є базовим для більшості алгоритмів інтерполяції, який потребує найменшого часу обробки, оскільки враховує тільки один піксель – найближчий до точки інтерполяції. В результаті кожен піксель просто стає більше.

Тобто для кожного пікселя кінцевого зображення вибирається один піксель вихідного, найбільш близький до його положення з урахуванням масштабування. Зокрема, якщо виконується збільшення в ціле число раз m, то одному пікселю вихідного зображення відповідають m^2 пікселів збільшеного зображення, які мають той же колір. Такий метод дає пікселізоване зображення при збільшенні і сильно зернисте зображення при зменшенні. Метод простий, при цьому зображення зберігає різкість кордонів, проте проявляється аліасинг (зокрема, діагоналі схожі на «драбинку» з квадратів).

Взагалі, якість і продуктивність будь-якого методу зменшення можна оцінити по відношенню кількості пікселів, які брали участь у формуванні кінцевого зображення, до числа пікселів в оригінальному документі. Чим більше це відношення, тим швидше за все алгоритм якісніше і повільніше. Ставлення, рівне одному, означає що як мінімум кожен піксель вихідного зображення зробив свій внесок в кінцеве. Але для просунутих методів воно може бути і більше одного. Наприклад, якщо ми зменшуємо зображення методом найближчого сусіда в 3 рази по кожній стороні, то це співвідношення дорівнює 1/9 (рис.1.1). Тобто велика частина вихідних пікселів ніяк не враховується.

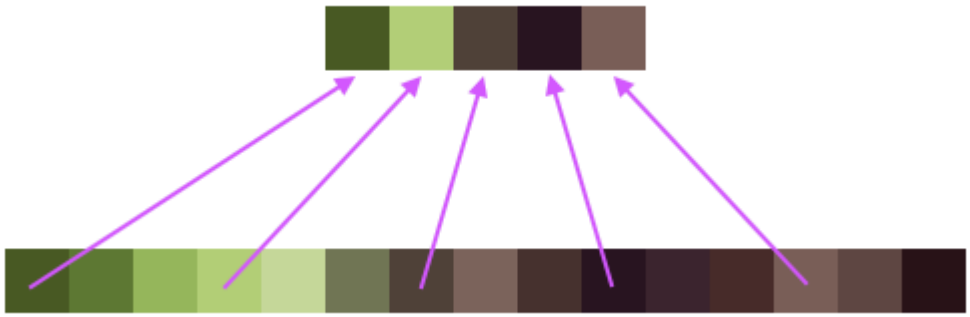


Рисунок 1.1 – Зменьшення зображення методом найближчого сусіда

Метод вкрай рідко застосовується для зменшення, тому що дає дуже погану якість, хоча і може бути корисний при збільшенні. Через швидкості і простоту реалізації він є у всіх бібліотеках та застосунках, що працюють з графікою.

1.2.1.2 Білінійна інтерполяція

В даний час є два види типових алгоритмів зміни масштабу зображення – білінійна і бікубічна інтерполяція.

Стандартні алгоритми масштабування, такі як білінійная і бікубічна інтерполяція, що використовуються для фотографій, абсолютно не підходять для растрового зображення – воно стає розмитим. Однак, існують методи, що збільшують чіткість зображень на великій здатності. Комп'ютери здатні виконувати ці алгоритми в режимі реального часу.

В данному випадку начення кольору в довільній точці представляється як лінійна комбінація кольорів в деякій околиці точки [1]:

$$\sum_{i,j} d_{ij} W(i-x) W(i-y), \tag{1.1}$$

де x, y – координати пікселя нового зображення, що обчислюється, в системі координат старого зображення.

Наприклад, нехай w,w^* – ширина старого і нового зображення відповідно. Якщо пікселі в матриці кольорів нумеруються з нуля, то координаті x^* в новому зображенні, відповідає координата $(w–1)t$ в старому, причому $t = x^*/(w^*–1)$. Очевидно, t завжди міститься на відрізку [0; 1]. Функція W визначає вагу пікселя і формалізує ставлення віддаленості пікселів: чим далі піксель від (x,y), тим менше його вага. Ваги поза околиці нульові.

Білінійна інтерполяція розглядає квадрат 2x2 відомих пікселів, що оточують невідомий. В якості результату інтерполяції використовується значення зваженого усереднення цих чотирьох пікселів. У результаті зображення виглядають значно більш гладко, ніж результат роботи методу найближчого соседа.

Функція ваги, яка використовується для (1.1), має вигляд (рис.1.2а):

$$W(x) = \begin{cases} 1-|x|, |x|<1 \\ 0, |x| \geq 1 \end{cases} \tag{1.2}$$

Білінійна інтерполяція – це розтягнення звичайної інтерполяції для функції двох змінних. Ідея полягає в тому, щоб лінійно інтерполювати в одному напрямку, а потім в перпендикулярному напрямку. Такий прийом став популярним в комп'ютерній графіці, але, під час масштабування цифрових зображень простежується висока пікселізація зображення.

Даний тип інтерполяції використовується для обчислення кольорів додаткових пікселів відносно основних, це дає можливість пом'якшувати переходи. Значенням функції в цьому випадку приймається забарвлення пікселя, а квадрат, створений 4 основними пікселями, вважається одиничним. У цього алгоритму є і мінуси, головний з яких полягає в тому, що при збільшенні в N раз зобаження розміром X на Y точок, буде отримане зображення розміром не NX на NY, а (N (X–1) +1) на (N (Y–1) +1) точок. А все через те, що для останньої точки початкового зображення не можна знайти пару, з якої можна було б провести інтерполювання.

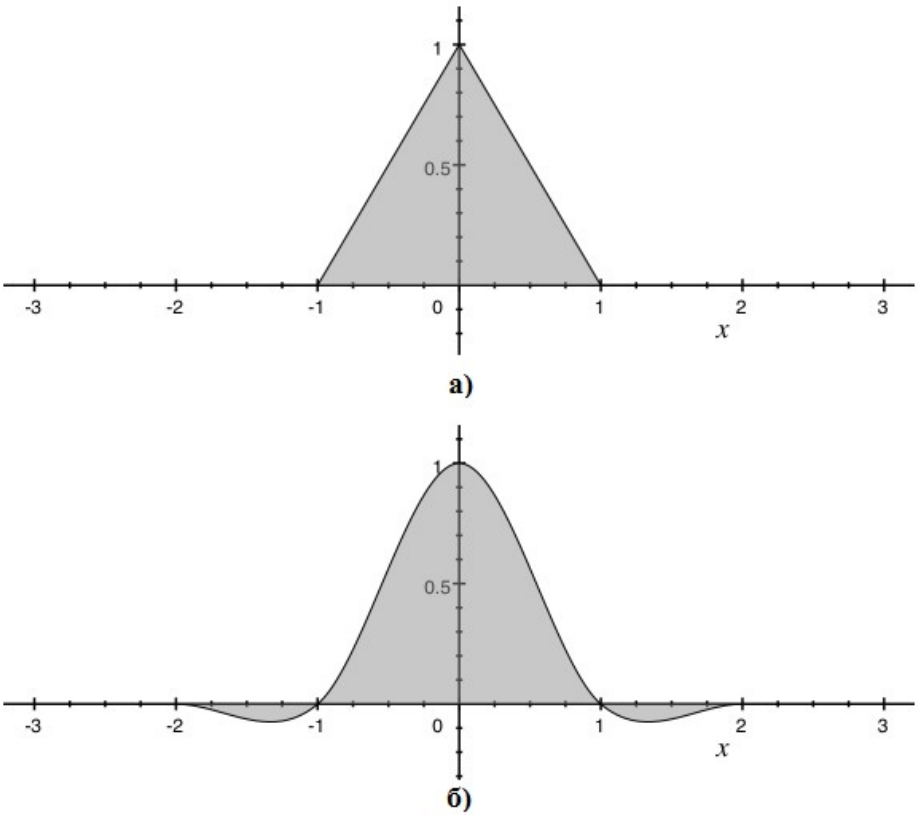


Рисунок 1.2 – Функції ваги пікселів: а – білінійна інтерполяція; б – бікубічна інтерполяція

1.2.1.3 Бікубічна інтерполяція

Бікубічна інтерполяція є природним продовженням білінійної і розглядає вже масив з 4x4 оточуючих пікселів – всього 16. Оскільки вони знаходяться на різних відстанях від невідомого пікселя, найближчі пікселі отримують при розрахунку більшу вагу. Бікубічна інтерполяція значно краще відтворює контрастні зображення, ніж попередні два методи, і можливо, є найбільш оптимальною по співвідношенню часу обробки і якості на виході. Тому вона стала стандартною для багатьох програм редагування зображень (наприклад, Adobe Photoshop), драйверів принтерів і вбудованої інтерполяції камер.

Функція ваги має вигляд (рис. 1.2б):

$$W(x)=\begin{cases}(x^2-1)(|x|-2)/2, & |x|<1\\ -(|x|-1)(|x|-2)(|x|-3), & 1<|x|<2\\ 0, \textit{інакше} & \end{cases} \qquad (1.3)$$

У бікубічної інтерполяції відбувається трохи інший процес, а саме збільшення кубічної інтерполяції для функції 2-х змінних, значення якої задані на 2D сітці. Результат кубічної інтерполяції – це гладка функція, а не поверхня, яку отримують під час білінійної інтерполяції або інтерполяції методом найближчого сусіда. Таким чином, бікубічна інтерполяція часто застосовують при обробці зображень, дозволяючи отримати більш якісне зображення відносно білінійної інтерполяції. Значення функції бікубічної інтерполяції в точці визначається через її значення в 16 найближчих точках.

Так як координати кольорів поточної точки часто обчислюються методом інтерполяції 4-ох найближчих, отримане зображення виходить розмитим. Не дивлячись на те, що це прийнятно для кольорових зображень, даний метод призводить до зменшення контрастності (різкості на контурах), і, тому, цей алгоритм дає погані результати для зображень з індексованою палітрою. Отже, обидва стандартних алгоритму мають недоліки.

Функції, що наведені в (1.2) і (1.3), є досить поширеними для визначення ваг пікселів.

Отже, ідеального методу для ресемплінгу растрової графіки необхідно інтерполювати зони суцільного тону, зберігати різкість горизонтальних і вертикальних ліній і пом'якшувати (за допомогою антиаліасингу) діагональні лінії і криві. Тобто всі розглянуті типові алгоритми будуть давати невірну початкову картинку. Отже, необхідний такий алгоритм масштабування

растрових зображень, який зможе об'єднати кілька позитивних ефектів від типових алгоритмів, і зведе до мінімуму їх недоліки.

1.2.1.4 Афінні перетворення

Афінні перетворення – загальний метод для масштабування зображень. Вони дозволяють за одну операцію повернути, розтягнути і відобразити зображення. Тому в багатьох програмах і бібліотеках, що реалізують метод афінних перетворень, функція зміни зображень є просто обгорткою, яка розраховує коефіцієнти для перетворення.

Принцип дії полягає в тому, що для кожної точки кінцевого зображення береться фіксований набір точок вихідного і інтерполюється відповідно до їх взаємного розташування і обраного фільтру. Кількість точок теж залежить від фільтра. Для білінійної інтерполяції береться 2x2 вихідних пікселя, для бікубічної – 4x4. Такий метод дає гладке зображення при збільшенні, але при зменшенні результат дуже схожий на найближчого сусіда. Фактично, аффінне перетворення не можна використовувати для зменшення більш ніж в 2 рази. І навіть при зменшенні до двох разів вони дають помітні ефекти драбинки для ліній.

Афінний перетворення можуть здійснювати поворот, переміщення і масштабування. На рис. 1.2 показано дію афінного перетворення на множині точок в R^2.

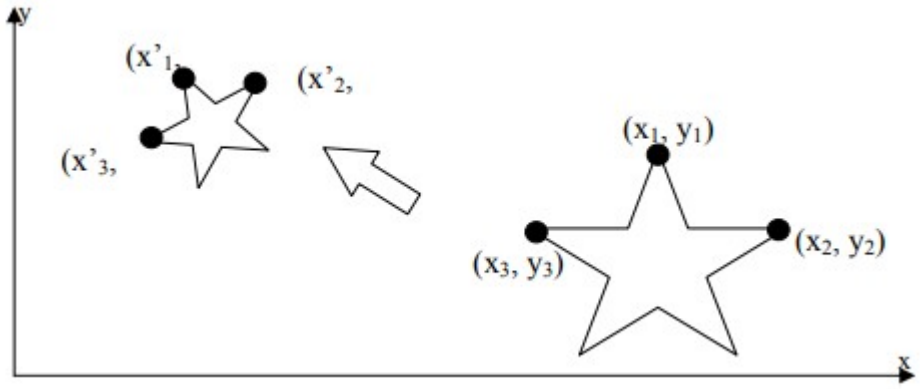


Рисунок 1.2 – Вплив афінного перетворення на точки в просторі R^2

Час роботи методу афінних перетворень помітно більше, ніж у найближчого сусіда, і залежить від розміру кінцевого зображення і розміру вікна обраного фільтра.

Метод знайшов широке застосування в програмах і бібліотеках. Цей спосіб використовується в усіх браузерах для методу канви drawImage(). Крім цього, такий метод використовується в OpenCV, поточної версії бібліотеки Pillow, в Paint.NET.

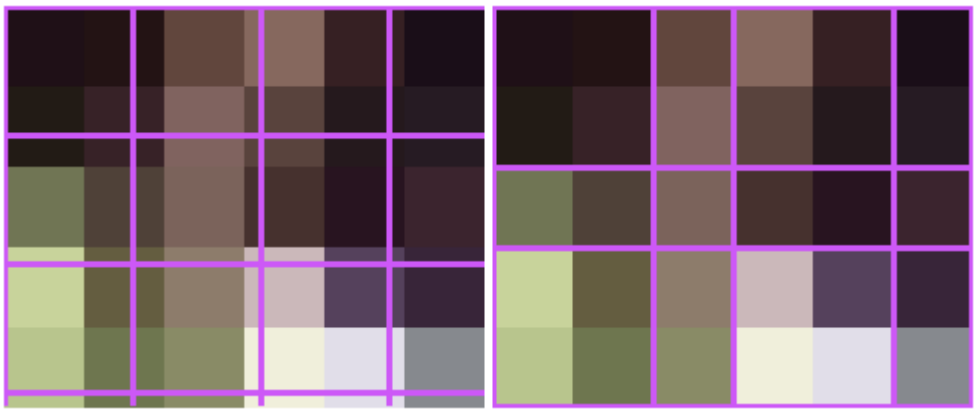
Крім того, саме цей метод використовується відеокартами для відтворення тривимірних сцен. Але різниця в тому, що відеокарти для кожної текстури заздалегідь готують набір зменшених версій (mip-рівнів), і для остаточного відтворення вибирається рівень з такими показниками, щоб зменшення текстури було не більше двох разів. Крім цього, для усунення різкого стрибка при зміні mip-рівня (коли текстурований об'єкт наближається або віддаляється), використовується лінійна інтерполяція між сусідніми mip-рівнями (це вже трилінійна фільтрація). Таким чином для відтворення кожного пікселя тривимірного об'єкту потрібно інтерполювати між 2^3 пікселями. Це дає прийнятний для швидко рухомої картинки результат за час, лінійний відносно кінцевого дозволу.

1.2.1.5 Метод Supersampling

За допомогою цього методу створюються mip-рівні та працює повноекранне згладжування в іграх. Суть методу у розділенні вихідного зображення за сіткою пікселів кінцевого і складанні всіх вихідних пікселів, що припадають на кожен піксель кінцевого, відповідно до площі, що потрапила під кінцевий піксель. При використанні цього методу для збільшення, на кожний піксель кінцевого зображення доводиться рівно один піксель вихідного. Тому результат для збільшення дорівнює методу найближчого сусіда.

Можна виділити два види цього методу: з округленням кордонів пікселів до найближчого цілого числа пікселів і без (рис.1.3). У першому випадку алгоритм стає малопридатним для масштабування менше ніж в 3 рази, тому що на який-небудь один кінцевий піксель може припадати один вихідний, а на сусідній – чотири (2x2), що призводить до диспропорції на локальному рівні. У той же час алгоритм з округленням очевидно можна використовувати у випадках, коли розмір початкового зображення кратний розміру кінцевого, або масштаб зменшення досить малий (версії з дозволом 330×220 майже не відрізняються). Відношення оброблених пікселів до вихідних при округленні кордонів завжди дорівнює одиниці.

Метод без округлення дає відмінну якість при зменшенні на будь-якому масштабі, а при збільшенні дає дивний ефект, коли велика частина вихідного пікселя на кінцевому зображенні виглядає однорідною, але на краях видно перехід. Відношення оброблених пікселів до вихідних без округлення кордонів може бути від одиниці до чотирьох, тому що кожен вихідний піксель вносить вклад або в один кінцевий, або в два сусідніх, або в чотири сусідніх пікселів.



а) б)

Рисунок 1.3 – Метод суперсемплінгу: а) сітка без округлення кордонів пікселів до найближчого цілого числа; б) сітка з округленням кордонів пікселів до найближчого цілого числа

Продуктивність цього методу для зменшення нижче, ніж у афінних перетворень, тому що в розрахунку кінцевого зображення беруть участь всі пікселі вихідного. Версія з округленням до найближчих кордонів зазвичай швидше в кілька разів. Також можливо створити окремі версії для масштабування в фіксовану кількість разів (наприклад, зменшення в 2 рази), які будуть ще швидше.

Даний метод використовується в функції gdImageCopyResampled() бібліотеки GD, що входить до складу PHP, є в OpenCV (прапор INTER_AREA), Intel IPP, AMD Framewave. Приблизно за таким же принципом працює libjpeg, коли відкриває зображення в зменшеному в кілька разів вигляді. Останнє дозволяє багатьом програмам відкривати зображення JPEG заздалегідь зменшеними в кілька разів без особливих накладних витрат.

1.2.1.6 Метод Convolution

Цей метод схожий на аффіннt перетворення тим, що використовуються фільтри, але має не фіксоване вікно, а вікно, пропорційне масштабу. Наприклад, якщо розмір вікна фільтру дорівнює 6, а розмір зображення зменшується в 2,5 рази, то у формуванні кожного пікселя кінцевого зображення бере участь $(2{,}5 \cdot 6)^2 = 225$ пікселів, що набагато більше, ніж в разі суперсемплінга (від 9 до 16). Згортки можна рахувати в 2 проходи, спочатку в одну сторону, потім в іншу, тому алгоритмічна складність розрахунку кожного пікселя дорівнює 225, а всього $(2{,}5 \cdot 6) \cdot 2 = 30$. Внесок кожного вихідного пікселя в кінцевий визначається фільтром. Відношення оброблених пікселів до вихідних цілком визначається розміром вікна фільтру і так само його квадрату. Тобто для білінійного фільтру це відношення буде 4, для бікубічного – 16, для Ланцоша – 36 (рис.1.4). Алгоритм прекрасно працює як для зменшення, так і для збільшення.

Швидкість роботи цього методу залежить від усіх параметрів: розмірів вихідного зображення, розміру кінцевого зображення, розміру вікна фільтра.

Саме цей метод реалізований в ImageMagick, GIMP, в поточній версії Pillow з прапором ANTIALIAS.

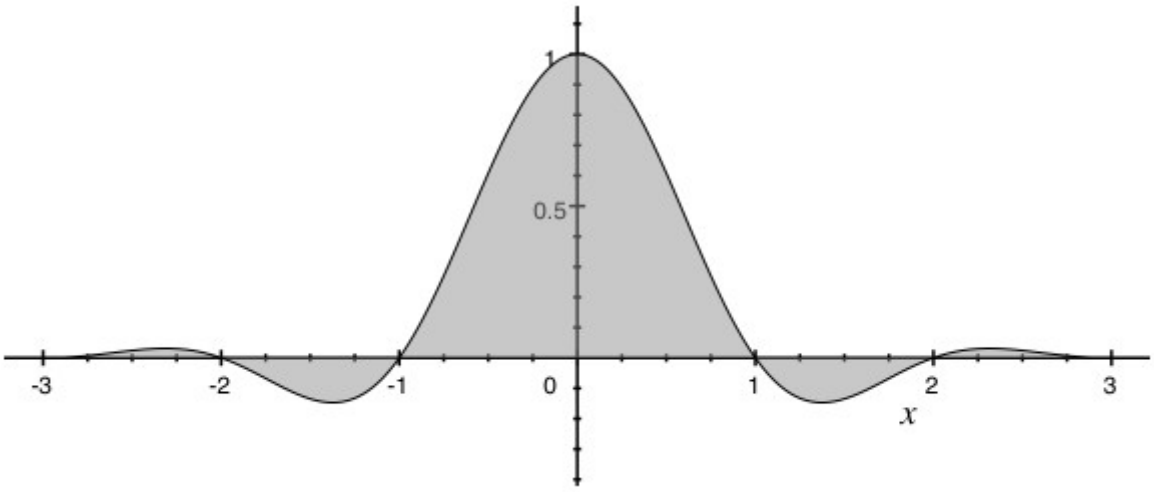


Рисунок 1.4 – Фільтр Ланцоша

Одна з переваг методу в тому, що фільтри можуть задаватися окремою функцією, ніяк не прив'язаної до реалізації методу. При цьому функція самого фільтра може бути досить складною без особливої втрати продуктивності, тому що коефіцієнти для всіх пікселів в одному стовпці і для

всіх пікселів в одному рядку розраховуються тільки один раз. Тобто сама функція фільтра викликається тільки $(m+ n) \cdot w$ раз, де m і n – розміри кінцевого зображення, а w – розмір вікна фільтру.

Деякі фільтри мають зони від'ємних коефіцієнтів (як наприклад бікубічний фільтр або фільтр Ланцоша). Це потрібно для додання переходам на кінцевому зображенні різкості, яка була на вихідному.

1.2.2 Адаптивні методи інтерполяції растрової графіки

Адаптивні алгоритми включають в себе багато комерційних алгоритмів в ліцензованих програмах, таких як Qmage, PhotoZoomPro, Geue Fratal і інші. Багато з них використовують різні варіанти своїх алгоритмів (в основному, на основі попіксельного аналізу), коли виявляють наявність кордону – з метою мінімізації недоліків інтерполяції в тих місцях, де вони найбільш помітні. Як правило, ці алгоритми в першу чергу призначені для задач збереження максимальної деталізації вихідних локальних ефектів в збільшених зображеннях. Несмотря на досить великий інтерес до задачі масштабування, жоден з підходів має недоліки, пов'язаніз трансформацією цифрового зображення з одної координатної сітки в іншу. Проведений аналіз методів масштабування зображень дозволив виявити основні причини недостатнього ефекту відомих підходів: використання детермінованих процедур інтерполяції стаціонарних процесів до апріорно нестаціонарних вихідних даних (характерно для неадаптівних алгоритмів); націленість процедур масштабування для вирішення локальних завдань перетворень (характерно для адаптивних алгоритмів).

1.3 Порівняльний аналіз алгоритмів масштабування растрової графіки

EPX. Огляд алгоритмів масштабування растрової графіки хотілося б почати з алгоритму Eric'sPixeleXpansion (EPX) або піксельне збільшення Еріка. Цей метод, був створений Еріком Джонстоном з LucasArts в 1992 році в процесі перенесення ядра SCUMM з IBM PC (дозвіл 320×200, 256 кольорів) на перші кольорові комп'ютери компанії Apple, з дозволом великим в два рази. Алгоритм можна описати таким чином. Припустимо, що дана матриця пікселів з 9 елементів і відомий центральний і бічні пікселі, а кутові невідомі, тоді порівнюючи бічні пікселі можна було отримати кутові, за однакової кількості 3 пікселів, за кутові брався центральний піксель.

Подальші реалізації цього методу (AdvMAME2x і Scale2x, створені в 2001 році) мають іншу (більш ефективну), але функціонально ідентичну,

реалізацію. Існують ще пара алгоритмів AdvMAME4x / Scale4x – це подвійний (два рази застосований) EPX. В свій час цей алгоритм вважався дуже ефективним і його часто використовували, але пізніше з'явилися ще більш вдосконалені алгоритми.

Окремо хочеться відзначити алгоритм Scale3x / AdvMAME3x. Цей алгоритм багато в чому схожий з EPX, а головною відмінністю можна назвати те, що в ньому оцінюється не 4 пікселя, а всі 9.

Eagle. Наступний метод ресемплінгу растрової графіки – Eagle. Принцип роботи алгоритму: для всіх точок на вході створюється 4 вихідних, спочатку всі крапки фарбуються в колір поточного пікселя (як і в найближчому сусідові). Потім беруться точки зверху і зліва, і, якщо вони одного кольору (всі три), то фарбуємо і ліву верхню точку в цей колір. Ці дії повторюються для всіх 4 точок, і виконується перехід до наступного набору точок. Тобто, в результаті, наприклад, одинична чорна точка на білому фоні після цього методу зникне. Дана помилка виправлена в методах2xSaI і HQ3.

Алгоритм депікселізації (Копфа-Лісчінскі). На конференції SIGGRAPH в серпні 2011 року співробітник Microsoft Research Йоханнес Копф спільно з професором Дані Лісчінскі представили дослідницьку роботу з описом нового алгоритму депікселізації.

Алгоритм призначений для перетворення 8-бітних зображень. При цьому він вирішує проблему масштабування пікселів і конвертує їх в області з різним ступенем згладжування, які строго розділені контурними лініями. В оригінальному зображенні сусідні пік селі по діагоналі пов'язані тільки 1 точкою, через це дрібні деталі при збільшенні стають візуально непов'язаними. Це породжує неточності при визначенні зв'язків діагональних сусідів.

Основними примітивами в алгоритмі є B-сплайн криві, які визначають згладжування контурів між областями. Розглянемо граф з (w+1) (h+1) вузлами, що представляє зображення розміром WH (рис.1.5). Кожен піксель відповідає замкнутій комірці графа. Горизонтальні і вертикальні сусідні комірки мають сусідні ребра в цьому графі, а діагональні сусіди поділяють тільки вершину. Ці діагональні сусіди стають візуально розділеними при масштабуванні, в той час як сусіди із загальним краєм залишаються візуально пов'язаними.

Рисунок 1.5 – Вихідне зображення 16x16 пікселей

Першим завданням алгоритму є змінити вихідні комірки таким чином, щоб кожна пара сусідніх пікселів, які повинні залишитися з'єднаними, відповідала тим коміркам, які мають загальне ребро.

Після зміни графа визначаться межі, де сусідні пікселі мають суттєво різні кольори. Ці межі ми помічаємо як видимі, тому що вони формують основу для видимих контурів в нашому остаточному векторному поданні, в той час як інші ребра що будуть безпосередньо видимими, оскільки будуть знаходитися в згладжених областях (рис.1.6).

Для отримання згладжених контурів квадратичні B-сплайн криві вписуються у послідовності видимих ребер. Це значно покращує плавність результату, однак зображення все ще має ступеневий ефект. Тому у подальшому відбувається оптимізація контрольних точок B-сплайн кривих. Оптимізація відбувається за рахунок пошуку мінімуму суми енергії кожного вузла:

$$argmin_{p_i} \sum_i E^{(i)}, \tag{1.4}$$

де p_i – розташування i-го вузла. Енергія вузла визначається як сума гладкості і позиції:

$$E^{(i)} = E_s^{(i)} + E_p^i. \tag{1.5}$$

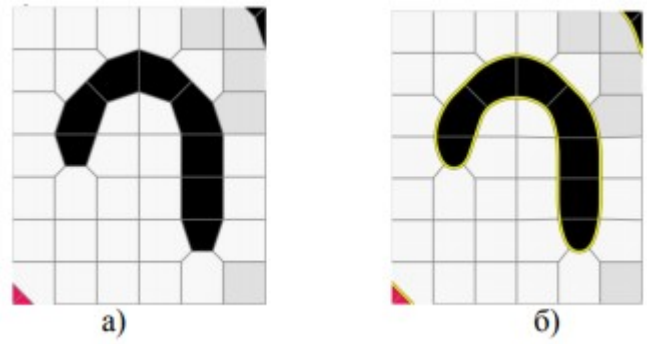


Рисунок 1.6 – Реконструкція пікселів комірок: а) зв'язки у вихідному графі; б) згладжені контури квадратичними B-сплайн кривими

Ці два терміни мають рівну частку в енергії. Гладкість визначається як відсутність кривини. Тому можна визначити енергію гладкості як:

$$E_s^{(i)} = \int_{s \epsilon r(i)} \left| k(s) \right| ds, \tag{1.7}$$

де *r(i)* – ділянка кривої, *k(s)* – кривина в точці *s*. Інтеграл обчислюється чисельно шляхом дискретизації кривої на фіксованому інтервалі.Для того щоб запобігти занадто сильної зміні об'єктів, потрібно додатково обмежити контрольні точки. Енергія позиції визначається як:

$$E_p^{(i)} = \left\| p_i - \hat{p}_i \right\|^4, \tag{1.8}$$

де $\hat{p_i}$ – початкова позиція і –го вузла.

Піднесення до четвертого степеня дозволяє вузлам переміщатися відносно вільно в межах їх первісної позіції.Тем не менше, через розташування кривих, контрольні точки сильно варіюються в зв'язку з низьким дозволом основної піксельної сітки, результат все ще може бути ступінчастим. Тому на останньому етапі оптимізуються криві лінії, щоб зменшити ступенчатість. Зображення відрисовується шляхом інтерполяції кольорів, з використанням радіальних базисних функцій [4], як це показано на рис.1.7.

Рисунок 1.7 – Результуюче зображення

Новий метод вимагає набагато менше ресурсів, робить обробку зображень в кілька разів швидше. Алгоритм працює з сучасними моніторами, які мають велику кадрову частоту. А це допоможе підняти консолі та комп'ютерні дисплеї на абсолютно новий рівень графіки.

Hqnx. Розглянемо сімейство алгоритмів hqnx, які застосовуються для масштабування. Максим Степін створив методи hq2x, hq3x і hq4x для ресемплінгу в пропорціях 2: 1, 3: 1 і 4: 1. Забарвлення кожної точки порівнюється з 8-ю сусідами, останні відзначаються як близькі й далекі, після цього застосовується перегенерована таблиця для знаходження потрібного відношення значень для всіх вихідних точок (4, 9 або 16). Метод hq3x чудово пом'якшує діагоналі з нахилом ±1: 2, ±1:1 і ±2:1 (якщо відсутня антиаліасінг на вході методу). Лінії з іншим множником нахилу представляються як ламані з попередніх ліній. Відмінно вирівнюються круті криві. На відміну від 2xSaI, у виведенні використовується антиаліасінг. Хочеться пояснити, що антиаліасінг це згладжування – метод, який використовується для усунення «драбинки», яка виникає на краях зображень (плоских або об'ємних), що виводяться в один момент часу на монітор. Вирівнювання придумали в 1972 році в Массачусетському технологічному інституті в ArchitectureMachineGroup, що стала в подальшому основою MediaLab. Спочатку hqnx використовувався для емулятора SuperNintendo, ZSNES.

Алгоритми ресемплінгу піксельної графіки застосовуються в емуляторах консолей. Швидкі комп'ютери при застосуванні цих методів дають можливість виведення зображення з іншим масштабом зі швидкістю додатків реального часу, наприклад, в іграх. Методи з високою оптимізацією видають ясне і чітке зображення з практично непомітним розмиванням. Методи ресемплінгу растрової графіки написані для величезної кількості емуляторів, 2D-ігрових, наприклад, для AdvanceMAME, DOSBox. Всі вони мають високі оцінки геймерів, які портируют і переписують гри, написані в

80-х і 90-х. На даний момент дані фільтри використовуються в сервісах XboxLive, VirtualConsole, і PSN для кращого відображення старих ігор на дисплеях з високою роздільною здатністю. Приклади таких ігор: Sonic's Ultimate Genesis Collection, Castlevania: The Dracula X Chronicles, Castlevania: Symphony of the Night, iAkumajō Dracula X Chino Rondo.

Зрозуміло, що розвиток алгоритмів масштабування піксельної графіки було можливо завдяки, в основному, використанню графічних зображень в комп'ютерних іграх. Раніше коли можливість обчислювальних машин була невеликою, то задовольнялися таким зображенням, яке було, збільшуючи зображення простими алгоритмами. Коли ж можливості почали возрастати, то захотілося зробити картинку більш привабливою для очей. Звичайних алгоритмів було недостатньо, довелося придумувати нові, більш ефективні.

Проведений оглядовий аналіз алгоритмів масштабування піксельної графікі показав, що кожний з них досить активно застосовувався свого часу, а деякі використовуються і до сих пір. Сказати однозначно який краще – не можна, тому що в різних ситуаціях ставляться різні вимоги до якості зображення, що масштабується. Однак, можна з впевненістю констатувати, що розглянуті новітні алгоритми мають одну відміну від класичних алгоритмів масштабування: вони формують згладжену, плавну картинку, візуально прибираючи ефект ступенчатості.

На сьогоднішній день триває пошук оптимального алгоритму масштабування. Головним критерієм роботи такого алгоритму можна назвати: якість формує мого зображення, мінімум обчислювальних ресурсів та швидкість отримання результату. Потужності комп'ютерів дозволяють придумувати складні алгоритми, залишилося знайти оптимальні методи. Пошук алгоритму масштабування входить в групу питань, які зараз викликають найбільший інтерес у фахівців.

2 ОПИС АЛГОРИТМУ КОНТЕНТНО-ЗАЛЕЖНОГО МАСШТАБУВАННЯ

2.1 Постановка проблеми

Різноманітність та універсальність пристроїв відображення на сьогодні висуває нових вимог до цифрових носіїв. Так, дизайнери повинні створювати різні варіанти веб-контенту та розробляти макети для різних пристроїв. Крім того, HTML, як і інші стандарти, може підтримувати динамічні зміни макету сторінки та тексту. Тим не менш, сучасні зображення, хоча вони є одним із ключових елементів цифрових носіїв, як правило, залишаються незмінними за розміром і не можуть масштабуватися, щоб автоматично підходити до різних макетів. Тобто розмір або співвідношення сторін зображення повинні змінюватися, повинні вписуватися в різні дисплеї, такі як мобільні телефони, з можливістю друкування їх із заданим розміром або роздільною здатністю. Стандартне масштабування зображення не є достатнім, оскільки воно погіршує зміст зображення, і зазвичай його можна застосовувати рівномірно. Обрізка обмежена, оскільки вона може видалити лише пікселі з периферії зображення.

Ефективне стиснення зображень повинне не тільки використовувати геометричні обмеження, але й враховувати вміст зображення. В роботі [5] авторами запропонований простий оператор масштабування зображень, що називається Seam carving, який підтримує зміну розміру зображень для вмісту як для зменшення, так і для розширення. Сім (seam) – оптимальний з'єднаний шлях пікселів на одному зображенні зверху вниз або зліва направо, де оптимальність визначається енергетичною функцією зображення, яка розраховує важливість пікселів. Тобто сім – це сполучений шлях з пікселями з низькою енергією. Завдяки багаторазовому вирізанню або вставці сімів у одному напрямку можна змінити співвідношення сторін зображення (рис.2.1). Застосовуючи ці оператори в обох напрямках, можна перетворити зображення до нового розміру. Вибір і порядок сімів захищають вміст зображення, як це визначено функцією енергії. При зменшенні зображення, вибір сімів гарантує, що при збереженні структури зображення буде вилучено більше пікселів з низькою енергією (рис.2.2). При збільшенні зображення вставки сімів забезпечує баланс між вихідним вмістом зображення та штучно вставленими пікселями. Фактично, ці оператори створюють змістовне змінення розмірів зображень.

Рисунок 2.1 – Зображення з одним горизонтальним і одним вертикальним сімом

а

б

Рисунок 2.2 – Результати масштабування: а – масштабування алгоритмом Seam carving; б – стандартне масштабування

Алгоритм Seam carving також може бути використаний для поліпшення вмісту зображення та видалення об'єкта. Зберігаючи порядок сімів у зображенні, можна створювати багаторозмірні зображення, здатні постійно змінюватися в режимі реального часу, щоб відповідати заданому розміру.

2.2 Поняття «сімів»

Seam carving дозволяє видалити пікселі з нецікавих частин зображення, зберігаючи при цьому важливий вміст. Seam carving може змінювати розмір зображення, граційно вирізаючи чи вставляючи пікселі в різні частини зображення. Цей алгоритм використовує функцію енергії, яка визначає, наскільки важливі пікселі [5].

«Сім» являє собою з’єднаний шлях пікселів, що мають низьку енергію, який перетинає зображення згори донизу чи зліва направо. «Сіми» можуть бути як вертикальні, так і горизонтальні. Вертикальний «сім» – це шлях з’єднаних пікселів, що проходять зверху донизу зображення, з одним пікселем в кожному рядку зображення. Горизонтальний «сім» схожий на вертикальний за винятком того, що з’єднання проходить зліва направо. Функція енергії, яка вказує на важливість пікселів, оцінює піксель, вимірюючи його контрастність з сусідніми пікселями.

Іншими словами, «сім» – це найкоротший шлях у графі, в якому вершинами є пікселі, а вага ребер дорівнює значенню функції енергії (рис. 2.3).

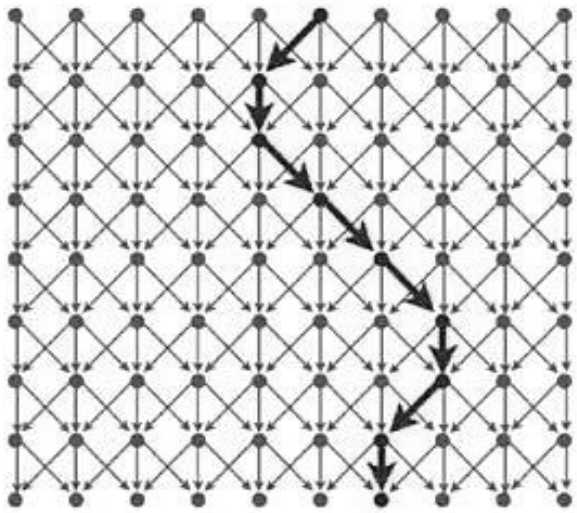


Рисунок 2.3 – Вертикальний «сім»

Формально, нехай I – це зображення n x m. Визначимо вертикальний «сім» як:

$$s^x=\{s_i^x\}_{i=1}^n=\{(x(i),i)\}_{i=1}^n,\ \forall i,|x(i)-x(i-1)|\le 1, \quad (2.1)$$

де x – відображення x: [1,…, n]→ [1,…, m]. Тобто вертикальний «сім» – це з'єднаний шлях пікселів в зображенні зверху вниз, який містить один і тільки один піксель в кожному рядку зображення. Тобто для кожного елемента (i,j), що належить сіму, наступний елемент може бути тільки (i+1,j-1), (i+1,j), (i+1,j+1). Аналогічно, якщо y – відображення y: [1, …, m]→ [1, …, n], тоді горизонтальний «сім» визначається так:

$$s^y=\{s_j^y\}_{j=1}^m=\{(j,y(j))\}_{j=1}^m,\ \forall j,|y(j)-y(j-1)|\le 1. \quad (2.2)$$

Пікселі шляху «сіма» s (наприклад, вертикальний «сім» s) тоді будуть:

$$I_s=\{I(s_i)\}_{i=1}^n=\{I(x(i),i)\}_{i=1}^n. \quad (2.3)$$

Послідовно видаляючи чи вставляючи «сіми», можемо як зменшити, так і збільшити розмір зображення в обох напрямках. Для зменшення зображення, вибір «сімів» гарантує, що зберігаючи структуру зображення, ми видаляємо більше пікселів з низькою енергією та менше – з великою енергією.

Для збільшення зображення, порядок вставки «сімів» забезпечує баланс між вмістом початкового зображення та пікселями, що були штучно вставлені. Ці оператори здійснюють, по суті, зміну розміру зображення з урахуванням вмісту.

Алгоритм Seam carving може використовуватися для зміни співвідношення сторін, зміни розміру зображення, підвищення вмісту зображення, видалення об'єктів, захисту об'єктів на зображенні від деформації під час зміни розміру.

2.3 Визначення інформативності змістовного наповнення зображення

Для знаходження ланцюжка пікселів з найменшою сумарною значимістю необхідно виділити на зображенні межі об'єктів – їх контури. Задача виділення контурів полягає в побудові додаткового зображення, яке

містить тільки контури зображення. Для цього зображення можна перетворити в чорно-біле того ж дозволу, в якому яскравість і колір кожного пікселя замінюється відповідним значенням відтінка сірого (від 1 до 256).

Як правило, межа предмета на фотографії відображається перепадом яскравості між двома порівняно однорідними областями. Однак перепад яскравості може бути пов'язаний з текстурою предмета, предметами, що відскакують тінями або, навпаки, відблисками на поверхнях об'єктів. Таким чином, виділення контурів чутливо до шуму. Наявність шуму може привести до появи помилкових контурів, які не є межами областей з постійною або повільно мінливою яскравістю. Видалення шуму слід виконувати так, що б зберегти важливі для подальшого семантичного опису/розпізнавання деталі зображення. Для шумозаглушення або згладжування використовуються різні методи, які добре працюють з певними видами шумів. Аналіз показує, що найбільш адекватними щодо практично одержуваних шумів є моделі адитивного гаусового шуму і імпульсного шуму [6].

Після згладжування проводиться виділення контурів, для чого використовуються детектори країв і методи зв'язування точок контуру. Сформувавши карту країв, можна перейти до пошуку зв'язкових ланцюжків пікселів, що мають мінімальні суми по карті значущості. Потім отримані зв'язані ланцюжки пікселів використовуються для контекстного масштабування.

2.3.1 Способи зниження шумів

Лінійне усереднення. Найпростіша ідея згладжування полягає в усередненні значень яскравості сусідніх пікселів. Береться так звана апертура фільтру – вікно (частина зображення), з яким працює фільтр. Апертура поступово пересувається по зображенню зліва направо і зверху вниз на один піксель. В результаті на наступному кроці фільтр працює з вікном, в якому знаходяться не тільки елементи вихідного зображення, а й елементи, що раніше були перетворені. Чим більше апертура, тим сильніше усереднення. Найпростіший варіант фільтрації – для заданого розміру апертури обчислювати середнє арифметичне значення всіх пікселів апертури навколо центрального пікселя, якому потім і присвоюється отримане значення. Цікава модифікація цього методу була розроблена де Хаан, який запропонував брати сусідні пікселі не підряд, а через один або два пікселя. Стверджується, що при

такому підході пригнічується низькочастотний шум, який візуально більш помітний, ніж високочастотний [7].

Медіанний фільтр. Даний спосіб грунтується на знаходженні медіани – середнього елемента, а не середнього арифметичного послідовності. Для цього елементи апертури упорядковуються. Наприклад, для апертури 3 × 3 потрібно впорядкувати 9 точок. Потім значення п'ятого елемента впорядкованої послідовності привласнюється центральному пікселу вікна фільтра.

Згладжування за допомогою гауссіана. Дискретне гауссово ядро згладжування (апертуру фільтра) можна отримати, побудувавши масив розміром (2k + 1) × (2k + 1), значення елемента (i, j) якого:

$$H_{ij} = \frac{1}{2\pi\sigma} e\left(-\frac{(i-k-1)^2+(j-k-1)^2}{2\sigma}\right), \quad (2.4)$$

де σ – це середньоквадратичне відхилення гауссіана, параметр, який задає ступінь розмиття;

k – коефіцієнт розмірності масиву.

Дане ядро згладжування утворює таке зважене середнє, для якого в центрі ядра вагові коефіцієнти пікселів набагато більше, ніж на його межах. Маска фільтра така, що центральний елемент маски має найбільше значення, він відповідає піку розподілу Гаусса. Значення інших елементів зменшуються в міру віддалення від центрального елемента, що відбувається відповідно до розподілу Гаусса. Наступний приклад демонструє маску фільтра Гаусса розміром 5×5 з среднеквадратическим відхиленням 0,5 [8]:

0.0000 0.0000 0.0002 0.0000 0.0000
0.0000 0.0113 0.0837 0.0113 0.0000
0.0002 0.0837 0.6187 0.0837 0.0002
0.0000 0.0113 0.0837 0.0113 0.0000
0.0000 0.0000 0.0002 0.0000 0.0000

2.3.2 Аналіз методів виділення контурів і побудови карти енергії

Під контуром зображення розуміють просторово протяжний розрив, перепад або стрибкоподібну зміну значень яскравості. Для надійного визначення точки контуру як точки, що знаходиться на межі перепаду яскравості, зміна яскравості, яка асоційована з цією точкою, має бути істотно більшою, ніж зміна яскравості в точці фону. Яскравість точки оцінюється по відношенню до сусідніх з нею точок, тобто аналізується локальна область

зображення. Тому визначення того, яке значення є істотним, а яке – ні, полягає у встановленні величини порога.

Одним з найбільш відповідних природі зображень способів виявлення контурів є диференціювання яскравості, яка розглядається як функція просторових координат [9].

Seam carving може підтримувати декілька типів функції енергії, такі як градієнтна (gradient magnitude), візуальна салієнтність (visual saliency), eye-gaze movement, ентропія та інші. Отримати карту енергії можна, використовуючи функції енергії, що розглядаються нижче.

Градієнтна величина (gradient magnitude). Існує багато методів вилучити непомітні пікселі з зображення. Перший та найголовніший метод полягає в призначенні енергії для кожного пікселя за допомогою градієнтного оператора (Собеля, Превитта, Робертса, Лапласіана) та обчисленні градієнта в обох напрямках x та y [2]. Знайти контур зображення зі значеннями яскравості $I(x,y)$ можна шляхом взяття часткових похідних $\mathrm{d}I/\mathrm{d}x$ і $\mathrm{d}I/\mathrm{d}y$, якими оцінюють швидкість зміни яскравості в напрямках x і y відповідно. На практиці потрібно виділяти контури, напрямок яких є довільним. З цією метою використовується модуль градієнта функції яскравості, тоді функція енергії визначається наступним чином:

$$e(I)=e(x,y)=\left|\frac{\partial I}{\partial x}\right|+\left|\frac{\partial I}{\partial y}\right|. \quad (2.5)$$

Для цифрових зображень замість часткових похідних беруться різницеві функції [10]:

$$\frac{dI(x,y)}{dx}\approx s_1(n_1,n_2)=I(n_1,n_2)-I(n_1-1,n_2),$$
$$\frac{dI(x,y)}{dy}\approx s_2(n_1,n_2)=I(n_1,n_2)-I(n_1,n_2-1).$$

Для збільшення швидкості обчислення модуля градієнта можна скористатися наступною функцією:

$$e(n_1,n_2)=\left|s_1(n_1,n_2)\right|+\left|s_2(n_1,n_2)\right|. \quad (2.6)$$

Розглянемо реалізацію обчислень енергії на прикладі. Нехай точки зображення розміром 3x3 характеризуються своєю яскравістю (рис.2.4а). Спочатку розрахуємо по модулю різницю яскравостей між пікселем і його

сусідами (праворуч і знизу), а потім обчислимо енергію цього пікселя як середнє арифметичне цих значень. Для виділеного пікселя (рис.2.4б) різниця між ним і сусідом праворуч – 8, сусідом знизу – 3. Середнє арифметичне – (8 + 3) / 2 = 5.5. Відкинемо дробову частину, тобто енергія виділеного пікселя дорівнює 5. Аналогічно розраховуються значення інших пікселів. При цьому пікселі, які крайні праворуч і крайні знизу матимуть тільки сусіда праворуч або знизу, тому для них значення різниці і буде середнім арифметичним. Для пікселя у правому-нижньому куту таких сусідів взагалі немає, тому просто приймемо його енергію за 0. В результаті отримаємо матрицю енергій пікселів, наведену на рис.2.4в.

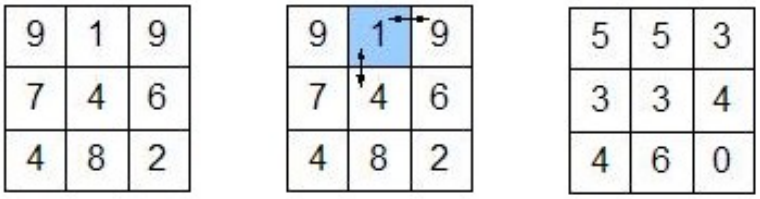


Рисунок 2.4 – Приклад побудови карти енергії градієнтним методом: а – вихідна матриця; б – розрахунок для виділеного пікселю; в – результуюча матриця

Якщо пікселі характеризуються не просто яркість, а значенням яркості окремо для R, G, B, то енергія пікселя дорівнює сумі енергій по кожній з компонент. Приклад карти енергій для зображення, наведений на рис.2.5.

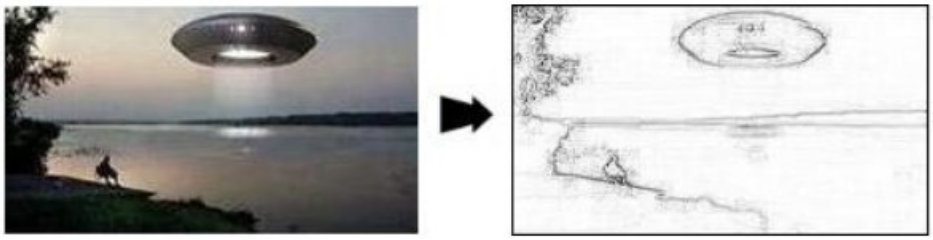


Рисунок 2.5 – Приклад карти енергій для зображення

Оператор Робертса. Найбільш загальним способом пошуку розривів яскравості є обробка зображення за допомогою ковзної маски, яку називають також фільтром, ядром, вікном або шаблоном, яка представляє собою якусь квадратну матрицю, відповідну зазначеній групі пікселів вихідного

зображення. Елементи матриці прийнято називати коефіцієнтами. Оперування такою матрицею в будь-яких локальних перетвореннях називається фільтрацією або просторової фільтрацією. Процес заснований на простом переміщенні маски фільтра від точки до точки зображення; в кожній точці (x, y) відгук фільтра обчислюється з використанням попередньо заданих зв'язків. У разі лінійної просторової фільтрації відгук задається сумою добутку коефіцієнтів фільтра на відповідні значення пікселів в області, покритої маскою фільтра.

Нехай область 3x3, що показана на рис. 2.6, являє собою значення яскравості в околиці деякого елемента зображення.

z_1	z_2	z_3
z_4	z_5	z_6
z_7	z_8	z_9

Рисунок 2.6 – Околиця 3x3 всередині зображення

Один з найпростіших способів знаходження перших приватних похідних в точці image полягає в застосуванні наступного перехресного градиентного оператора Робертса [1]:

$$E_x = z_9 - z_5 \tag{2.7}$$

$$E_y = z_8 - z_6 \tag{2.8}$$

Ці похідні можуть бути реалізовані шляхом обробки всього зображення за допомогою оператора, описуваного масками на рис.2.7, використовуючи процедуру фільтрації, описану раніше.

-1	0
0	1

0	-1
1	0

Рисунок 2.7 – Маски оператора Робертса

Реалізація масок розмірами 2x2 не надто зручна, тому що у них немає чітко вираженого центрального елемента, що істотно відбивається на

результаті виконання фільтрації. Але цей «мінус» породжує дуже корисну властивість даного алгоритму – високу швидкість обробки зображення.

Оператори Собеля і Превитта. За своєю суттю ці оператори є дискретними диференціальними операторами, які обчислюють наближене значення градієнта яскравості зображення. Для цього оператори використовують маски 3×3, за допомогою яких згортають вихідне зображення для обчислення наближених значень похідних по горизонталі і вертикалі (рис.2.8). У оператора Собеля (рис.2.9) вплив шуму кутових елементів дещо менше, ніж у оператора Превитта (рис.2.8), що істотно при роботі з похідними. У кожної з масок сума коефіцієнтів дорівнює нулю, т. е. ці оператори будуть давати нульовий відгук в областях з постійною яскравістю.

Наведені маски використовуються для виділення горизонталей і вертикалей. В операторі Собела виводиться ваговий коефіцієнт 2 для середніх елементів. Таке збільшене значення зменшує ефект згладжування за рахунок надання більшої ваги середніх точок.

-1	-1	-1
0	0	0
1	1	1

-1	0	1
-1	0	1
-1	0	1

Рисунок 2.8 – Маски Превитта

-1	-2	-1
0	0	0
1	2	1

-1	0	1
-2	0	2
-1	0	1

Рисунок 2.9 – Маски Собела

Для область 3x3, показаному на рис.2.6, що являє собою значення яскравості в околиці деякого елемента зображення, оператор Превитта задається виразами:

$$E_x=\left(z_7+z_8+z_9\right)-\left(z_1+z_2+z_3\right) \qquad (2.9)$$

$$E_y=(z_3+z_6+z_9)-(z_1+z_4+z_7) \quad (2.10)$$

Відповідно оператор Собеля задається виразами:

$$E_x=(z_7+2z_8+z_9)-(z_1+2z_2+z_3) \quad (2.11)$$

$$E_y=(z_3+2z_6+z_9)-(z_1+2z_4+z_7) \quad (2.12)$$

Лапласіан. Недоліком виділення контурів за допомогою градієнта є отримання "широких" контурів – шириною в декілька пікселів. Для виділення тонкого контуру в один піксель використовується лапласіан двовимірної функції I(x,y).

$$\Delta^2 I(x,y)=\frac{\partial^2 I(x,y)}{\partial x^2}+\frac{\partial^2 I(x,y)}{\partial y^2}. \quad (2.13)$$

Оскільки зображення представляється дискретним набором пікселів, оператор Лапласа можна реалізувати як процедуру лінійної обробки зображення апертурою 3×3 – три рядки з номерами 0, 1, 2 і три стовпці – 0, 1, 2. Цифрова апроксимація може бути представлена у вигляді:

$$\partial^2 I=4n_{11}-(n_{01}+n_{10}+n_{12}+n_{21}). \quad (2.14)$$

Лапласіан застосовується для визначення, на якій стороні контуру – темній або світлій – знаходиться розглянутий піксель. До недоліків Лапласіан можна віднести дуже високу чутливість до шумів, можливість появи розривів в контурі і виникнення помилкових контурів. Тому разом з гауссовским згладжуванням дуже часто застосовується лапласіан. Гауссовское згладжування і пошук Лапласіан можна виконувати одночасно, що скорочує час обробки зображення. Оскільки взяття другої похідної є лінійної операцією, згортка зображення з маскою Лапласіан еквівалентна тому, як якщо б зображення спочатку згорталося з гаусою згладжу вальною функцією, а потім обчислювався лапласіан результату. Апроксимуюча маска для отримання згортки зображення з лапласіаном гауссіана має вигляд [8]:

$$\begin{matrix} 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ -1 & -2 & 16 & -2 & -1 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \end{matrix}$$

Ентропія. Базова градієнта функція не дає певних результатів. Фільтр локальної ентропії може бути застосований на зображення для того, щоб поліпшити базову градієнтну карту енергії [11].

Гістограма градієнтів (HoG). Функція енергії визначається наступним чином:

$$e_{HoG}(I)=\frac{\left|\frac{\partial I}{\partial x}\right|+\left|\frac{\partial I}{\partial y}\right|}{max(HoG(I(x,y)))} \quad (2.15)$$

де $HoG(I(x,y))$ береться як гістограма орієнтованих градієнтів в кожному пікселі. Таким чином, взяття максимом HoG в знаменнику притягує «сіми» до країв зображення, в той час, як чисельник гарантує, що «сім» буде проходити паралельно краю та не буде його перетинати.

Жодна з розглянутих функцій енергії не працює добре на всіх зображеннях, проте, в цілому, вони всі підходять однаковому діапазону для зміни розміру зображення. Вони відрізняються в швидкості, з якою вони встановлюють візуальні артефакти та частини зображення, на які вони впливають.

Градієнтна величина та гістограма градієнтів працюють однаково добре. Надалі розглядається градієнтна функція енергії.

2.3.3 Алгоритм знаходження ланцюжка пікселів з мінімальною сумою градієнта

Маючи функцію енергії, припустимо, що потрібно зменшити ширину зображення. Для досягнення цього можна придумати декілька стратегій. Наприклад, оптимальна стратегія збереження енергії (тобто збереження пікселів з великим значенням енергії) буде полягати в видаленні пікселів з найнижчою енергією в зростаючому порядку. Це зруйнує прямокутну форму зображення, тому що ми можемо видалити різну кількість пікселів з кожного рядка. Якщо ми хочемо запобігти руйнування зображення, можна видалити

однакову кількість пікселів з низькою енергією з кожного рядка. Це збереже прямокутну форму зображення, проте зруйнує вміст зображення, створюючи ефект зигзагів. Щоб зберегти як форму, так і візуальну узгодженість зображення, можна використовувати автообрізку. Тобто, потрібно знайти підвікно цільового розміру зображення, яке містить найвищу енергію. Ще одна можлива стратегія, щось між видаленням пікселів та обрізкою, полягає в видаленні цілих стовбців з найнижчою енергією. Однак, тоді можуть з'явитися артефакти в кінцевому зображенні. Таким чином, потрібен оператор зміни розміру зображення, який буде менш обмежувальним, ніж обрізка чи видалення стовбців, але зможе зберегти контент зображення краще, ніж видалення окремих пікселів. Це призводить до стратегії Seam carving та визначенню внутрішніх «сімів».

Треба зауважити, що аналогічно видаленню цілого рядка чи стовбця зображення, видалення пікселів «сіма» з зображення має тільки локальний ефект: усі пікселі зображення зсуваються вліво (чи вгору) для компенсації шляху, якого бракує. Візуальний ефект помітний лише уздовж шляху «сіма», залишаючи іншу частину зображення без змін. Також зауважимо, що можна замінити обмеження $|x(i)-x(i-1)|\le 1$ в (2.1) на $|x(i)-x(i-1)|\le k$, та отримати або простий стовбець (чи рядок) при $k=0$, або кусково-з'єднаний чи навіть повністю роз'єднаний набір пікселів для будь-яких значень $1\le k\le m$.

Маючи функцію енергії *e*, можна визначити вартість «сіма» як:

$$E(s)=E(I_s)=\sum_{i=1}^{n} e(I(s_i)). \tag{2.16}$$

Треба знайти оптимальний «сім» s^{i}, який мінімізує витрати на цей «сім»:

$$s^{i}=\min_{s} E(s)=\min_{s}\sum_{i=1}^{n} e(I(s_i)). \tag{2.17}$$

Оптимальний «сім» може бути знайдений за допомогою динамічного програмування. Для цього на основі матриці градієнтів, в якій кожен елемент має значення градієнта відповідного пікселя, розраховується матриця сум за алгоритмом:

− перший рядок аналогічний першому рядку матриці градієнтів;

− другий і всі наступні рядки формуються однаково – береться значення градієнта відповідного пікселя і складається з найменшим значенням градієнта

трьох сусідніх елементів попереднього рядки матриці. Таким чином буде обчислена кумулятивна мінімальна енергія M для всіх можливих з'єднаних «сімів» для всіх (i,j).

Алгоритм може бути формалізований наступним чином:

$$\begin{cases} e(i,j), i=0 \\ M(i,j)=e(i,j)+¿min(M(i-1,j-1),M(i-1,j),M(i-1,j+1)), i>0 \end{cases} \quad (2.18)$$

Ланцюжки закінчуються пікселями, розташованими в останній рядку. Пошук ланцюжка починається з пікселя в останньому рядку, для якого аналізуються три сусідніх з наступного рядка і вибирається мінімальний. Потім знайдений елемент стає основою для наступного кроку і т. д. Знайдені ланцюжка з мінімальною сумою можуть вилучатися з зображення або додаватися до нього без зміни розмірів основних об'єктів зображення, так як точки їх контурів, що мають великі значення, в ланцюжка не потраплять і важливі структурні властивості зображення будуть збережені.

Визначення M для горизонтальних «сімів» аналогічне: мінімальне значення в останньому стовбці в M буде вказувати кінець мінімального з'єднаного горизонтального «сіма».

2.4 Зміна розмірів зображення

Припустимо, що ми хочемо змінити співвідношення сторін заданого зображення I розміру n x m наn x m', де $m-m'=c$. Це може бути досягнуто шляхом простого послідовного видалення c вертикальних «сімів» з I. Всупереч звичайному масштабуванню, цей оператор не буде змінювати важливі частини зображення (визначаються функцією енергії), і в дійсності змінює розмір зображення, враховуючи вміст. Така ж корекція відношення сторін, від n x m доn x m' також може бути досягнута, збільшуючи кількість рядків на коефіцієнт m/m'. Додана вартість такого підходу полягає в тому, він не видаляє ніяку інформацію з зображення.

Щоб збільшити зображення, треба вставити нові штучні «сіми» до зображення. Отже, для того, щоб збільшити розмір зображення I, потрібно знайти оптимальний вертикальний (горизонтальний) «сім» s на зображенні I та дублювати пікселі s, усереднюючи їх між лівими та правими сусідами (верхніми та нижніми у горизонтальному випадку). Будемо позначати

результуюче зображення $I^{(-1)}$. Повторення цього процесу скоріш за все створить розтягування артефакту, обираючи один і той самий «сім».

Щоб досягнути ефективного збільшення, важливо балансувати між контентом початкового зображення та частинами, що були штучно вставлені. Таким чином, щоб збільшити зображення на k, треба знайти перші k «сімів» для видалення (тобто, що мають найменшу енергію) та дублювати їх, щоб отримати $I^{(-k)}$. Це можна розглядати як процес переміщення назад в часі, щоб відновити пікселі більшого зображення, які були б видалені шляхом видалення «сімів».

Дублювання всіх «сімів» в зображенні еквівалентно стандартному масштабуванню. Щоб продовжити враховувати вміст зображення при надмірному збільшенні (наприклад, більш ніж на 50%), потрібно розбити процес на кілька кроків. Кожен крок не збільшує розмір зображення більш, ніж на величину збільшення розміру з попереднього кроку, тим самим істотно зберігаючи важливий контент зображення від розтягування. Тим не менш, надмірне збільшення зображення, найбільш імовірно, виробить помітні артефакти.

2.5 Посилення контенту зображення. Видалення та захист об’єктів

Посилення контенту зображення. Замість збільшення розміру зображення, seam carving може бути використаний для посилення вмісту зображення при збереженні його розміру. Цього можна досягнути шляхом комбінування масштабування та seam carving. Щоб зберегти вміст зображення якомога більше, треба спочатку використати стандартне масштабування, щоб збільшити зображення і тільки потім застосувати seam carving для більшого зображення, щоб видалити зайві «сіми» та повернути зображення до початкового розміру. Пікселі, які видалені, є фактично суб-пікселями початкового зображення.

Видалення об’єктів. Для видалення об’єктів користувачі повинні виділити цільовий об’єкт, який потрібно видалити. Пікселям цього об’єкту призначається мінімальна енергія. Потім видаляються «сіми» із зображення, поки не зникнуть усі виділені пікселі. Можна також відновити початковий розмір зображення, застосувавши вставку «сімів» до отриманого (меншого) зображення. Треба зауважити, що на відміну від попередніх технік видалення об’єктів (Bertalmio [13], Criminski [14], Drori [15]), ця схема змінює усе зображення (або його розмір, або його вміст). Це відбувається тому, що як

видалені, так і вставлені «сіми» можуть проходити в будь-якому місці зображення.

Захист об’єктів. В деяких зображеннях «області інтересу» спотворюються та, таким чином, функція зміни розміру зображення, що враховує вміст, не працює. В таких випадках можна виділити області на зображенні, що нас цікавлять, та потім застосувати seam carving таким чином, що виділена область буде захищена, тобто мати найвищий пріоритет. Це досягається шляхом призначення пікселям виділеної області максимальної енергії.

Захист об’єктів можна також застосувати, якщо основні об’єкти на зображенні розділені довгим, порожнім простором. Виділивши їх, цей простір можна скоротити так, щоб об’єкти стали близькими один до одного.

3 ОПИС РОЗРОБЛЕНОГО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ТА РЕЗУЛЬТАТІВ

3.1 Структура розробленого програмного забезпечення

Для розробки програмного забезпечення обрано мову програмування – C# [16]. Середа розробки – Visual Studio Community 2015 [17]. Засіб розробки інтерфейсу – WPF.

Діаграма класів модуля редагування зображень наведено на рис. 3.1.

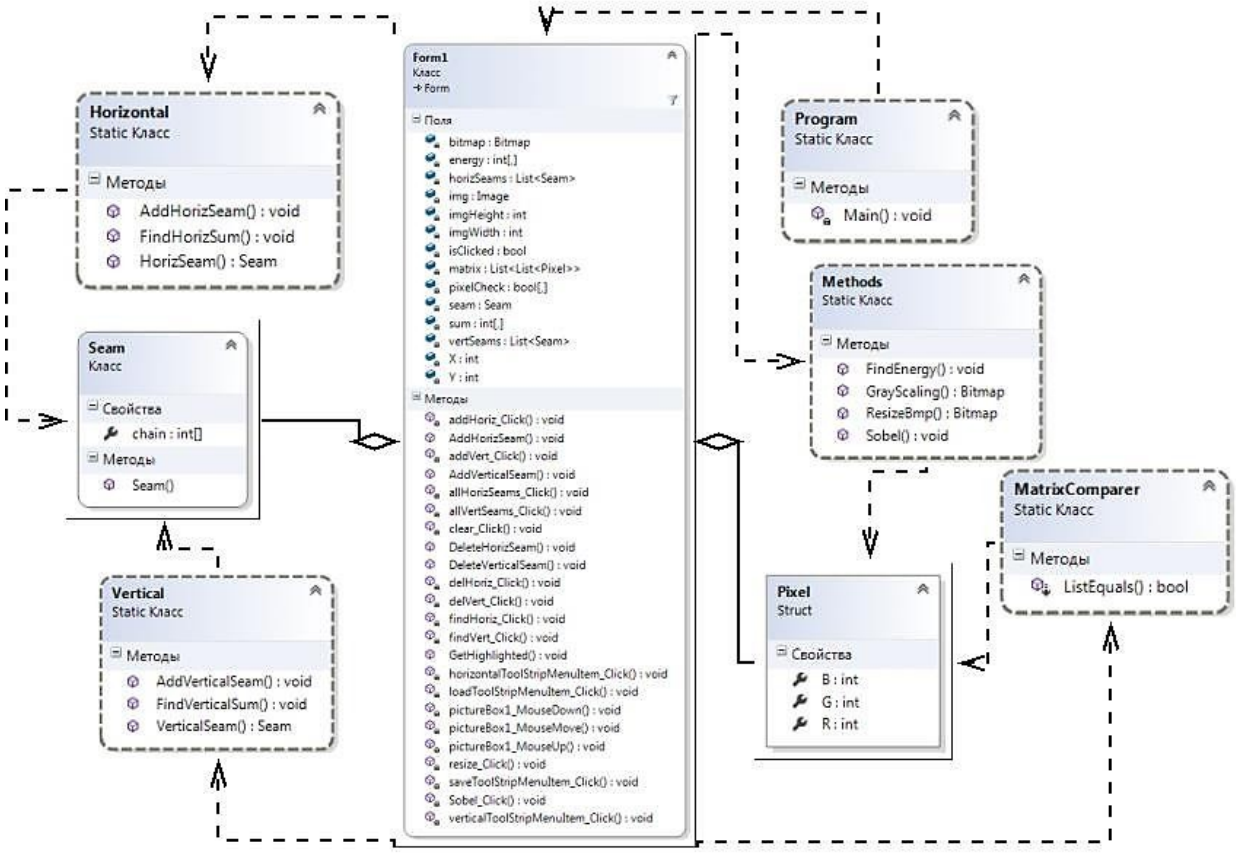


Рисунок 3.1 – Діаграма класів модуля редагування зображень

Розроблена програма складається з двох окремих модулів:

1) Систематизація зображень – каталогізація за змістовним наповненням зображень, пошук дублікатів, пошук схожих зображень, визначення найбільш відповідного каталогу при додаванні нових файлів.

2) Редагування зображень – збільшення та зменшення розміру зображення, враховуючи його зміст, видалення зайвих об’єктів, захист важливих об’єктів від деформації під час зміни розміру зображення,

обчислення градієнтних функцій енергії та оператора Собеля [12] з побудовою карти енергії на його основі.

Модуль редагування зображень містить 7 класів. Клас Seam призначений для створення об'єкту, що інкапсулює властивості окремого сіму. Для побудови горизонтального і вертикального шляху сімів створені статичні класи Horizontal та Vertical, які містять методи, що дозволяють знайти наступний у шляху сім і повернути його як об'єкт класу Seam.

Для реалізації алгоритму знаходження карти енергії пікселів з використанням градієнтного значення FindEnergy() та оператору Собеля Sobel() створені спеціальні методи, що знаходяться у статичному класі Methods. Шлях сімів представлений в програмі у вигляді зв'язаного списку, що містить об'єкти типу Seam. Для кольорового зображення створена структура для збереження значень каналів R, G і B.

Компоненти графічного інтерфейсу користувача представліні на формі Form1, з запуску якої починає свою роботу програмний модуль.

3.2 Опис програмної реалізації алгоритмів редагування зображень

Розглянемо більш докладно реалізації окремих алгоритмів масштабування зображень, що були використані в програмі.

Метод FindEnergy(). В главі 2 був розглянутий алгоритм знаходження енергії пікселя за градієнтним методом, суть якого зводиться до наступного. Спочатку розрахуємо по модулю різницю яскравостей між пікселем і його сусідами (праворуч і знизу), а потім обчислюється енергія цього пікселя як середнє арифметичне цих значень. Аналогічно розраховуються значення інших пікселів. При цьому пікселі, які крайні праворуч і крайні знизу матимуть тільки сусіда праворуч або знизу, тому для них значення різниці і буде середнім арифметичним. Для пікселя у правому-нижньому куту таких сусідів взагалі немає, тому просто приймемо його енергію за 0. Якщо пікселі характеризуються не просто яркість, а значенням яркості окремо для R, G, B, то енергія пікселя дорівнює сумі енергій по кожній з компонент.

Метод, що був реалізований в програмі працює з глобальними змінними: energy – масив, в який записуються енергія і image – масив, в якому зберігається зображення.

Блок-схема алгоритму наведена на рис. 3.2, код реалізації методу представлений у додатку А.

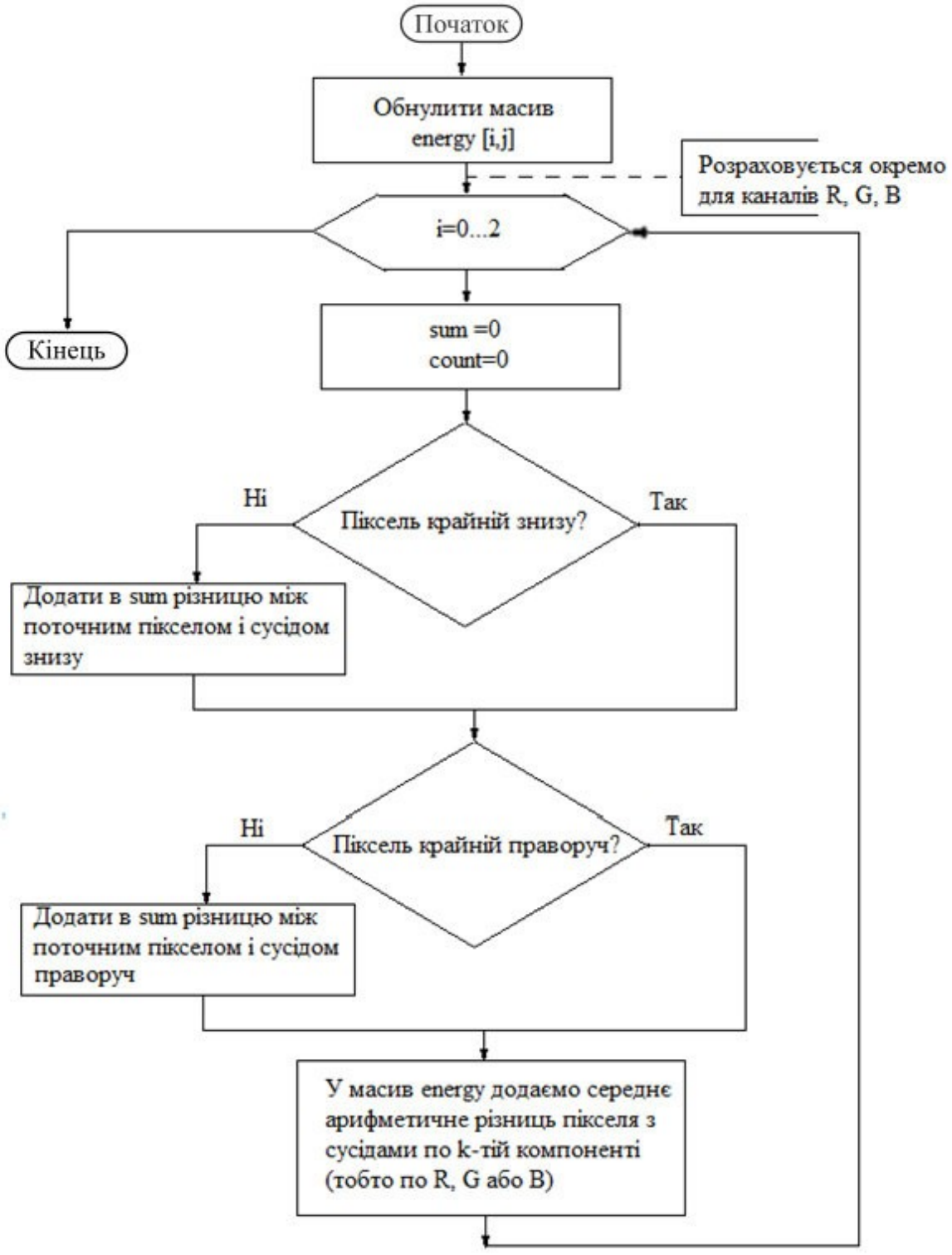


Рисунок 3.2 – Блок-схема алгоритму, реалізованому у методі FindEnergy()

Метод FindVerticalSum(). Після знаходження значення енергій для кожного пікселя, треба вибрати ті пікселі у яких значення енергії мінімальне. Але якщо почати забирати/додавати довільні пікселі, то зображення розповзеться, і деформується до невпізнання. Звичайно такий результат не

підходить. Тому спочатку треба вибрати ланцюжок пікселів, а потім вже їх видаляти або додавати. При чому недовільний ланцюжок, а правильний. Правильний ланцюжок – це набір точок, такий, що в кожному рядку зображення вибрано рівно 1 піксель, а пікселі в сусідніх рядках «з'єднані» або сторонами, або кутами.

Згідно алгоритму пропонують вибрати для видалення/розтягнення ті ланцюжки, з мінімальною сумою енергій пікселів, що входять до неї.

Перебір всіх ланцюжків, тобто код повного перебору навіть для невеликого зображення буде виконуватися дуже повільно. Краще використати динамічне програмування. Спочатку створити новий масив, який за розміром дорівнює масиву з енергіями пікселів. В цей масив для кожного пікселя запишемо суму елементів мінімального ланцюжка пікселів, який починається біля верхнього краю зображення і закінчується на даному пікселі.

Покажемо процес обчислення на прикладі. Нехай є масив зі значеннями енергій для кожного пікселя (рис. 3.3). Будемо обчислювати елементи цього масиву по рядках – від першого до останнього. Для верхнього рядка елементи даного масиву будуть рівні елементів масиву енергій (рис.3.3), оскільки з кожного такого елемента можна побудувати тільки 1 ланцюжок (всі одиничної довжини).

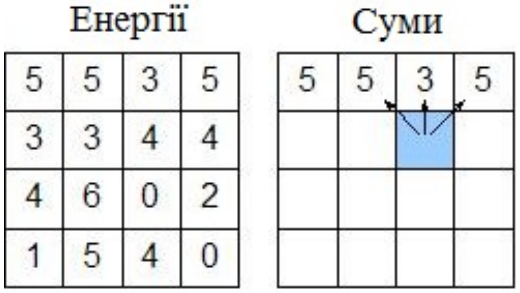


Рисунок 3.3 – Знаходження пікселей з мінімальною енергією (крок 1)

Далі обчислимо другий рядок. Розглянемо виділену клітинку. Щоб побудувати ланцюжок від цієї комірки, можна використати три способа, як показано на рис.3.4. З цих трьох способів треба вибрати мінімальний (оскільки треба розрахувати суми саме мінімальних ланцюжків). Для виділеної комірки це буде ланцюжок з сумою 3, і до цього ланцюжка додаємо енергію самої комірки. Тому в цій комірці запишемо число 7 (сума 3 і 4). Аналогічно розрахуємо всі суми для всіх елементів другого рядка (рис.3.4).

Третій рядок розраховується аналогічно другому. Знову розглянемо виділену клітинку. Від неї можна утворити такі ланцюжки:

- ця комірка плюс мінімальний ланцюжок довжини 8;
- ця комірка плюс мінімальний ланцюжок довжини 6;
- ця комірка плюс мінімальний ланцюжок довжини 7;

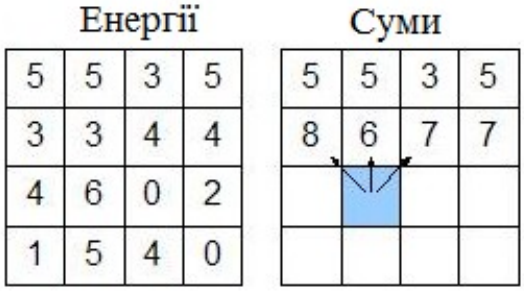


Рисунок 3.4 – Знаходження пікселей з мінімальною енергією (крок 2)

З цих варіантів знову вибираємо мінімальний (6) і додаємо до енергії самої комірки (6). Отримуємо значення цієї комірки – 12. Аналогічно розраховуємо інші елементи третього рядка.

Після обчислення таким чином всіх рядків отримаємо в результаті масив, представлений на рис. 3.5.

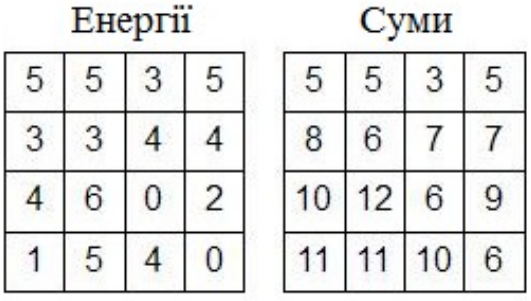


Рисунок 3.5 – Результат роботи алгоритму

Якщо формалізувати обчислення цього масиву, то отримаємо наступну формулу:

$$s[i,j]=\begin{cases} e[i,j] \; if \; i=0 \\ e[i,j]+min\left(s[i-1,j-1], s([i-1,j]), s([i-1,j+1])\right) \; if \; i\neq 0 \end{cases}$$

де *s* – масив сум, а *e* – масив енергій.

Варіант реалізації обчислення масиву сумм, наведений на рис. 3.6.

Алгоритм пошуку ланцюжка. По отриманому масиві можемо знайти ланцюжок з мінімальною сумою енергій. Спочатку знайдемо який піксель з нижньої строчки зображення належить цьому ланцюжку: елемент, що ми шукаємо, матиме найменше значення в масиві сум серед елементів нижньої строчки. Відповідно для всього зображення мінімальний ланцюжок буде обраний як мінімальний з усіх мінімальних ланцюжків, які закінчуються на пікселях в нижньому рядку (рис.3.7).

```
      private void FindVerticalSum() {
      // Для верхнього рядка значення sum і energy будуть співпадати
      for (int j = 0; j < imgWidth; j++)
          sum[0, j] = energy[0, j];
      // Для всіх інших пікселів значення елемента (i,j) масиву sum
      //дорівнюють        energy[i,j]+MIN(sum[i-1,j-1],sum[i-1,j],sum[i-
1,j+1])
      for (int i = 1; i < imgHeight; i++)
      for (int j = 0; j < imgWidth; j++)
      {
        sum[i,j] = sum[i-1, j];
        if (j>0 && sum[i-1, j-1] < sum[i,j]) sum[i,j] = sum[i-1, j-1];
           if (j<imgWidth-1 && sum[i-1, j+1] < sum[i,j]) sum[i,j] =
sum[i-
              1, j+1]; sum[i,j] += energy[i,j];
                }
          }
```

Рисунок 3.6 – Лістинг коду для метода FindVerticalSum()

Енергії				Сума			
5	5	3	5	5	5	3	5
3	3	4	4	8	6	7	7
4	6	0	2	10	12	6	9
1	5	4	0	11	11	10	6

Рисунок 3.7 – Пошук ланцюжка сімів (крок 1)

Ми вже знаємо на якому пікселі закінчується мінімальний ланцюжок, тепер можемо знайти піксель з передостанньої сходинки. Як вже було написано, з нижнього пікселя можемо піти тільки в 3 сусідні пікселі на рядок

вище: зліва-зверху, зверху або праворуч-зверху. Серед цих пікселів вибираємо піксель з мінімальним значенням в масиві сум, і переходимо до нього. Продовжуємо, поки не дійдемо до верхнього рядка. Процес показаний на наступному рис.3.8. Після виконання всіх цих операцій отримаємо набір пікселів, які можна змінити з мінімальними втратами для зображення. Пробна реалізація алгоритма наведена у додатку А.

Зменшення зображень. Після знаходження ланцюжка пікселів можна використовувати його для зміни розміру зображення. Збільшення і зменшення зображення трохи відрізняються, тому спочатку розглянемо зменшення, а потім вже збільшення.

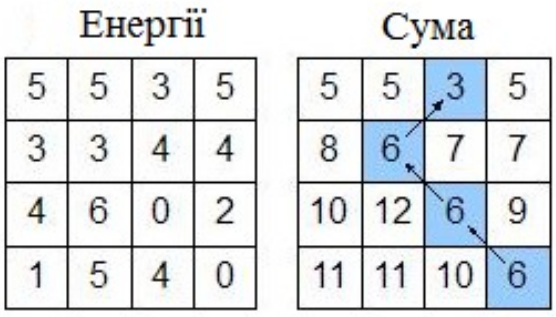


Рисунок 3.8 – Пошук ланцюжка сімів (фінальний етап)

Якщо нам треба зменшити ширину картинки на 1 піксель, то знаходимо вертикальний ланцюжок, як це описується вище, і видаляємо його з зображення. Цю операцію можна реалізувати за допомогою наступного циклу, представленому на рис. 3.9.

```
for (int i = 0; i <imgHeight; i ++)
   for (int j = cropPixels [i]; j <imgWidth; j ++)
   {
     pImage [i, j] = pImage [i, j + 1];
   }
```

Рисунок 3.9 – Цикл, що зменшує ширину зображення на 1 піксель

В *i*-му елементі масиву cropPixels записаний номер стовпчика пікселя, який видаляємо з *i*-го рядка. А сам цикл робить наступне: всі пікселі, які записані праворуч від тих, яких видаляємо, зрушуємо на одну точку вліво. В результаті цієї операції наш ланцюжок пікселів буде видалений з зображення.

Якщо треба зменшити зображення не на 1 піксель, а на більше значення, то виконуємо операцію видалення ланцюжка стільки разів, скільки треба (кожен раз при цьому треба буде цей ланцюжок шукати).

Хочу звернути увагу, що операція зменшення зображення на *n* пікселів і операція зменшення зображення на 1 піксель *n* раз не є еквівалентними. Їх відмінність в тому, що якщо ми зменшуємо на 1 піксель *n* раз, то для кожної з проміжних зображень шукаємо енергію пікселів, а якщо відразу зменшуємо на *n*, то енергії пікселів беремо з вихідного зображення.

3.3 Результати роботи модуля редагування зображень

В програмі реалізовано алгоритм seam carving, що представляє собою підхід зміни розміру зображення, що усвідомлено видаляє чи додає пікселі неважливих частин зображення.

Приклад результату пошуку оптимальних сімів наведено на рис. 3.9.

Рисунок 3.9 – Знайдені вертикальні та горизонтальні сіми

Можна також подивитися на карту енергії, яка реалізована за допомогою оператора Собеля (рис. 3.10)

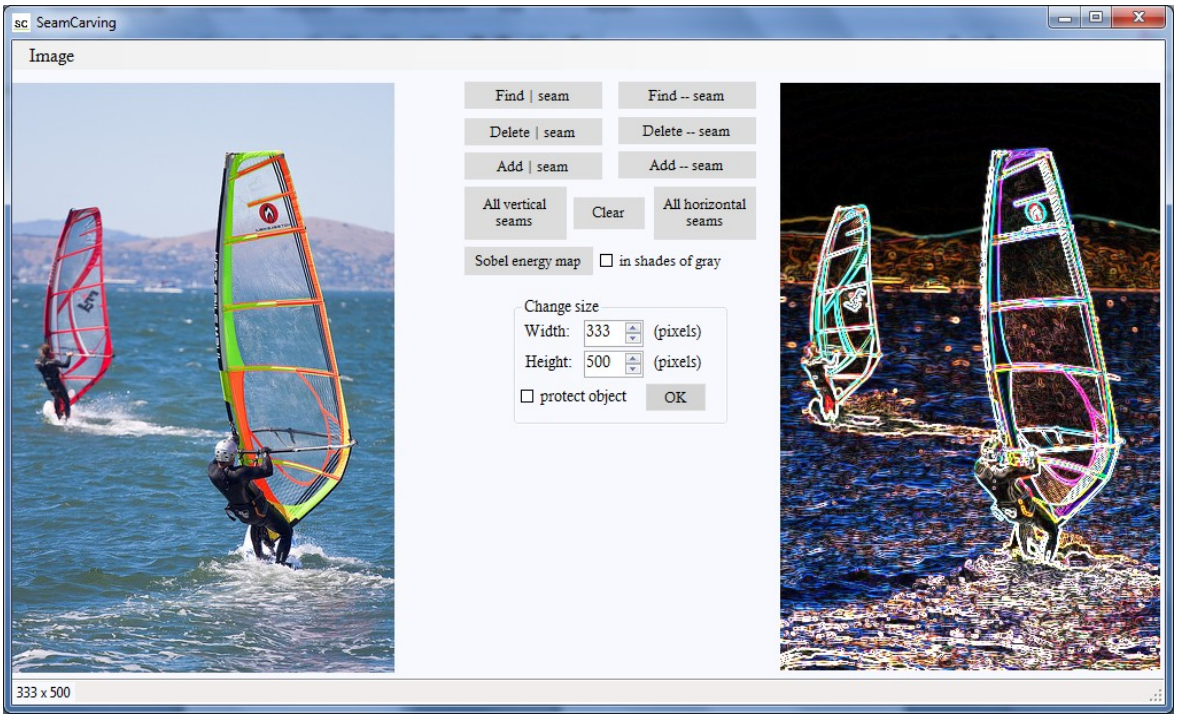


Рисунок 3.10 – Карта енергії Собеля

Приклад результату зменшення ширини зображення наведено на рис. 3.11.

Рисунок 3.11 – Початкове зображення та зображення, зменшене за шириною

Результат зменшення висоти зображення наведено на рис. 3.12.

Рисунок 3.12 – Початкове зображення та зображення, зменшене за висотою

Видно, що дошки серфінгістів при зменшенні розміру зображення деформувалися, оскільки алгоритм Seam carving порахував їх неважливими.

Система передбачає можливість виділення користувачем важливих областей для захисту від деформації під час зміну розміру (рис. 3.13).

Розглянутий підхід seam carving також застосовується для видалення об’єктів з зображення. Реалізовано наступну обчислювальну схему:

1) Будується енергетична карта зображення.
2) Визначається кумулятивна карта енергії зображення.
3) Встановлюються мінімальні значення енергії для M(i,j), якщо (i,j) – це обраний користувачем піксель.
4) Для кожної ітерації:

− визначається оптимальний «сім» (в потрібному напрямку – горизонтальний чи вертикальний);

− видаляється оптимальний «сім» із зображення, тим самим зменшуючи розмір зображення.

Приклади результату видалення об’єктів наведено на рис. 3.14.

Рисунок 3.13 – Виділення важливих об’єктів та зображення з незмінними об’єктами, що були виділені, зменшене за висотою

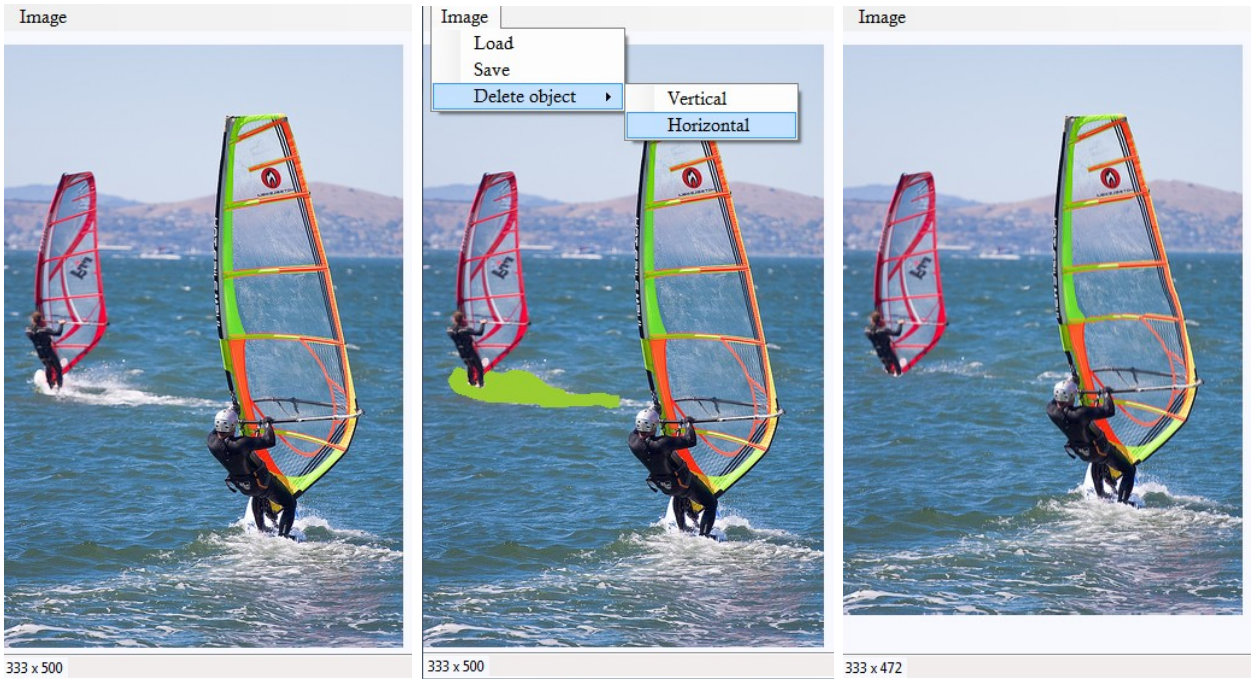


Рисунок 3.14 – Результат видалення виділеного об’єкту

Розроблена програма може працювати з різними форматами зображень (.jpg, .bmp, .png), а також зчитувати файли з піддиректорій, якщо вони є. Якщо в каталозі окрім зображень знаходяться інші файли, це не вплине на роботу програми.

ВИСНОВКИ

У магістерській роботі був виконаний аналіз та програмна реалізація алгоритму модифікації зображень з урахуванням їх змістовного наповнення. Запропонована в роботі модифікація належить до групи алгоритмів контентно-залежного масштабування і дозволяє змінювати розмір зображення за рахунок пошуку сімів (швів) – найменш важливих пікселів для подальшого їх видалення чи дублювання. Запропонований алгоритм базується на методі Seam carving, який використовує функцію енергії, що визначає важливість пікселей. Окрім зміни розміру цей підхід застосовується і для видалення зайвих об’єктів, визначених користувачем.

Магістерська робота складається з трьох розділів.

У першому розділі виконаний аналіз існуючих методів модифікації та масштабування зображень. Розглянуті особливості та властивості комп'ютерного зображення в його цифровому поданні. Надані класифікація та порівняльний аналіз роботи деяких методів масштабування растрової графіки. Показано, що найбільш популярними методами збільшення масштабу зображення є методи, засновані на інтерполяції кольорів пікселів. Принцип дії полягає в тому, що для кожної точки кінцевого зображення береться фіксований набір точок вихідного і інтерполюється відповідно до його взаємного розташування і обраного фільтра. В роботі здебільшого були розглянуті неадаптивні алгоритми інтерполяції: метод найближчого сусіда, білінійної інтерполяції, бікубічної інтерполяції та інші. Чем більше інтерполяційне вікно, що використовується, тим більш точним є перетворення цифрового зображення, але за рахунок значного збільшення часу обробки. Проведений оглядовий аналіз показав, що кожний з алгоритмів інтерполяції має певні недоліки, які пов’язані з втратаю якості зображення та наявністю артефактів.

У другому розділі роботи наведене обґрунтування вибору методу контентно-залежного масштабування Seam carving для вирішення проблеми модифікації зображень. Надані методологічні основи запропонованого методу. Розроблено обчислювальні схеми модифікацій методу для зменшення та збільшення розміру зображення з найменшими втратами інформації та захистом важливих об’єктів від деформації, а також видалення зайвих об’єктів. Для визначення інформативності пікселів зображення реалізовано обчислення градієнтних функцій енергії та оператора Собеля.

У третьому розділі роботи виконано проектування програмного забезпечення. Розроблена діаграма класів модуля редагування зображення і блок-схеми окремих програмно-реалізованих етапів процесу модифікації зображень. Для розробки програмного забезпечення обрано мову програмування – C# . Середа розробки – Visual Studio Community 2015. Засіб розробки інтерфейсу – WPF. Розроблена програма може працювати з різними форматами зображень (.jpg, .bmp, .png).

Тестування розробленого програмного забезпечення і аналіз отриманих результатів показав, що всі реалізовані функції: зменшення та збільшення розміру зображення (пропорційно, горизонтально, вертикально) , а також видалення зайвих об’єктів виконуються з найменшими втратами інформації та захистом важливих об’єктів від деформації.

ПЕРЕЛІК ПОСИЛАНЬ

1. Гонсалес Р., Вудс Р. Цифровая обработка изображений.// Пер. с англ.– М.: Техносфера. – 2006. –1072 с.
2. Ватолин Д., Ратушняк А., Смирнов М. Юкин В. Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео// – М.: ДиалогМИФИ. – 2003. – 384 с.
3. Сэломон Д. Сжатие данных, изображений и звука// – М.: Техносфера, 2004. – 368 с.
4. Трифанов А.И. Реализация собственного метода визуализации водной поверхности «скользящая текстура» / А.И. Трифанов, О.Ф. Абрамова // Современные наукоёмкие технологии. – 2013. – No 8 (ч. 1). – С. 96 – 97.
5. S. Avidan, A. Shamir. Seam carving for content-aware image resizing./ACM Transactions on Graphics (TOG) – Proceedings of ACM SIGGRAPH, Volume 26 Issue 3, July 2007. P. 13–21
6. Perona P., Malik J. Scale-space and edge detection using anisotropic diffusion// IEEE Trans. Pattern Anal. March.Intell. – 1990. – Vol. 12, No. 5. – P. 629–639.
7. Hanno S. Optimal operators in digital image processing: Дис. – Германия, 2000. [Електроний ресурс]. – URL: http://archiv.ub.uniheidelberg.de/ volltextserver/ volltexte/2000/962/
8. Обработка сигналов и изображений: Image Proc. Toolbox. [Електроний ресурс]. – URL: http://matlab.exponenta.ru/imageprocess/book/11/fspecial.php
9. Титов И. О., Емельянов Г. М. Выделение контуров изображения движущегося объекта // Вестник Новгородского гос. ун-та. – 2010. – № 55. – С. 27–31.
10. Гонзалес Р., Вудс Р. Цифровая обработка изображений. – М.: Техносфера, 2012. – 1104 с.
11. Рязанова Н.Ю., Ульихин В.А. Вопросы масштабирования изображений с учетом их содержания// Вестник МГТУ им. Н.Э. Баумана. Сер. “Приборостроение” – 2012. – С. 220–227.
12. Sobel I. History and Definition of the Sobel Operator. [Електроний ресурс].– URL: https://www.researchgate.net/publication/ 239398674_An_Isotropic_3x3_ Image_Gradient_Operator

13. Bertalmio M., Vese L., Sapiro G., and Osher S. Simultaneous structure and texture image inpainting / Proc. Conf. Comp. Vision Pattern Rec., Madison., 2003. – P. 707-714.
14. Criminisi A., Perez P., Toyama K. Region filling and object removal by exemplar-based inpainting / IEEE Transactions on Image Processing, 2004. – P. 417-424.
15. Drori I., Cohen-Or D., and Yeshurun H. Fragment-based image completion / ACM Trans. on Graphics SIGGRAPH, San Diego, US, 2003. – P.303-312.
16. Вагнер, Билл C# Эффективное программирование / Билл Вагнер. – М.: ЛОРИ, 2017. – 320 с.
17. Ник, Рендольф Visual Studio 2010 для профессионалов / Рендольф Ник. – М.: Диалектика /Вильямс, 2016. – 516 с