HCMUS - VNUHCM / FIT /

Computer Vision & Cognitive Cybernetics Department

Digital Image and Video Processing Application

Student ID: 21127690

Student name: Ngo Nguyen Thanh Thanh

Report: Morphological Operations for Grayscale Image

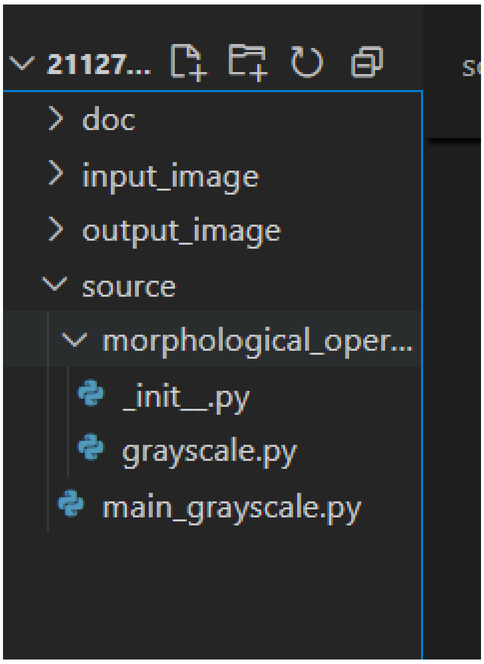
I. Evaluation summary:

No	Types	Task	Implemetation (Without OpenCV)	Implementation (With OpenCV)	Completion (%)
1	Gray scale	Grayscale Dilation	Implemented using max filter over neighborhood	Uses cv2.dilate()	100%
2		Grayscale Erosion	Implemented using min filter over neighborhood	Uses cv2.erode()	100%
3		Grayscale Opening	Implemented using grayscale erosion followed by dilation	Uses cv2.morphologyEx(…, cv2.MORPH_OPEN)	100%
4		Grayscale Closing	Implemented using grayscale dilation followed by erosion	Uses cv2.morphologyEx(…, cv2.MORPH_CLOSE)	100%
5		Grayscale smoothing	Implemented using Gaussian or median filtering		100%
6		Grayscale Morphology Gradient	Implemented using difference between dilation and erosion	Uses cv2.morphologyEx(…, cv2.MORPH_GRADIENT)	100%
7		Top-hat transformation	Implemented using difference between original image and opened image	Uses cv2.morphologyEx(…, cv2.MORPH_TOPHAT)	100%
8		Textural segmentation	Implemented using morphological filtering and thresholding		100%
9		Granulometry	Implemented using morphological opening with increasing kernel sizes		100%
10		Reconstruction	Implemented using marker-controlled		100%

			dilation until convergence		

II. List of features and file structure:

<u>File structure:</u>



input_image/ – Stores input images.

output_image/ – Contains processed output images.

source/ – Main source code directory.

- **morphological_operations/** – Contains morphological processing modules.
  - **grayscale.py** – Handles grayscale image operations.
- **main_grayscale.py** – Executes grayscale image processing.

Functions and methods used:

1. grayscale.py

Main functions:
pad_image_grayscale(img, kernel, pad_value) – Adds constant padding (value = 0 for dilation, 255 for erosion) to preserve output size and prevent border errors during processing.

• **grayscale_dilation(img, kernel**) – Applies grayscale dilation ($f \oplus b$) to brighten dark regions by computing the local maximum of (image + kernel).

• **grayscale_erosion(img, kernel)** – Applies grayscale erosion ($f \ominus b$) to darken bright regions by computing the local minimum of (image - kernel).

• **grayscale_opening(img, kernel)** – Performs grayscale opening ($f \circ b$) by erosion followed by dilation; removes small bright noise and smooths object boundaries.

• **grayscale_closing(img, kernel)** – Performs grayscale closing (f • b) by dilation followed by erosion; fills small dark gaps and connects adjacent objects.

• **grayscale_smoothing(img, kernel)** – Smooths image by applying opening then closing; suppresses both small bright and dark noise.

• **grayscale_morphology_gradient(img, kernel)** – Computes morphological gradient: highlights object boundaries by calculating (dilation - erosion).

• **top_hat(img, kernel)** – Extracts small bright features using Top-Hat transformation: (original - opening).

• **textural_segmentation(img, kernel)** – Performs texture-based segmentation using Top-Hat to highlight fine bright texture elements.

• **granulometry(img, sizes)** – Analyzes object size distribution by applying opening with multiple structuring element sizes and summing pixel intensities.

• **reconstruction(marker, mask, kernel, max_iter=100)** – Performs morphological reconstruction by iterative dilation of marker under mask constraint; useful for restoring image structures.

• **create_structuring_element(size)** – Creates a square structuring element (kernel) of a specified size.

2. main_grayscale.py

Main Functions:

- Load and preprocess the input image as a grayscale image from the specified file path.
- Provide two processing modes for morphological operations:
    - Manual: Implements grayscale morphological operations using custom algorithms defined in grayscale.py.
    - OpenCV: Uses built-in OpenCV functions for standard morphological processing.
- Support morphological operations:
  Original, Dilate, Erode, Open, Close, Smoothing, Gradient, TopHat, TexturalSegmentation, Granulometry, Reconstruction.
- Display and save output images using OpenCV and write granulometry results to text/plot files.
- Measure and report execution time for performance comparison between manual and OpenCV implementations.

Function Descriptions:

- **apply_manual(img, kernel, seed=(10,10))**
  Applies custom grayscale morphological operations using implementations from grayscale.py. Returns a dictionary containing operation names and their corresponding processed images. Also includes granulometry values (as a list) and image reconstruction results.
- **apply_opencv(img, kernel, seed=(10,10))**
  Applies standard grayscale morphological operations using OpenCV. Returns a dictionary of images corresponding to each operation.

- **operator(in_file, out_file, mor_op, mode, wait_key_time=5000)**
    - Loads and normalizes the input grayscale image.
    - Applies the selected morphological operation (mor_op) in the chosen mode (manual or opencv).
    - If no specific operation is provided, applies and displays all operations in a grid layout.
    - Special handling **for Granulometry**: outputs numerical values to .txt and saves a corresponding line plot image.
    - Measures execution time and displays results.
- **main(argv)**
    - Parses command-line arguments using getopt.
    - Accepts the following arguments:
        - -i / --in_file: Input image file path
        - -o / --out_file: Output file path
        - -p / --mor_operator: (Optional) Specific morphological operation to perform
        - -m / --mode: Processing mode (manual or opencv)
        - -t / --wait_key_time: Display wait time (in milliseconds)
    - Calls operator() with parsed parameters to execute the desired morphological processing workflow.

How to run code:

1. Install Required Libraries

First, make sure you have the necessary libraries installed, such as opencv, numpy, and morphological_operator. You can install them using pip:

pip install opencv-python numpy

Note: The morphological_operator library appears to be a custom-written library, so make sure it exists in the same directory or has been correctly installed.

2. Command Line Structure

Here's the command to run the program from the command line:

python main.py -i <input_file> -o <output_file> [-p <morph_operator>] -m <mode> -t <wait_key_time>

3. Explanation of Parameters:

- -i <input_file>: Path to the input image file.
- -o <output_file>: Path to save the result.
- -p <morph_operator> (Optional): The specific morphological operation you want to apply (e.g., "Dilate", "Erode", etc.).

- -m <mode>: The execution mode ("manual" for custom-written algorithms or "opencv" for OpenCV).
- -t <wait_key_time> (Optional): Time to wait (in milliseconds) before closing the window displaying the image.

4. Example:

Suppose you have an image file input.jpg and you want to apply the "Dilate" operation using the custom algorithm (manual mode) and save the result to output.jpg. You would run:

python main_graysacle.py -i input.jpg -o output.jpg -p “Dilate” -m manual

5. Options:

- If you don't specify a morphological operation, the program will apply all available operations.
- If you don't specify a wait time, the program will not wait and will automatically close the window displaying the image.

6. Error Information:

If there's an error, such as a missing image file or incorrect parameter, the program will show an appropriate error message.

Image proof:

Grayscale.py

source > morphological_operator > grayscale.py > ...

```python
import numpy as np

def pad_image_grayscale(img, kernel, pad_value):
    """Thêm padding với giá trị tùy theo phép toán (0 cho dilation, 255 cho erosion)."""
    pad_h, pad_w = kernel.shape[0] // 2, kernel.shape[1] // 2
    return np.pad(img, ((pad_h, pad_h), (pad_w, pad_w)), mode='constant', constant_values=pad_value)

def grayscale_dilation(img, kernel):
    """Grayscale Dilation: (f ⊕ b)(s,t) = max{f(s-x, t-y) + b(x,y)}"""
    padded_img = pad_image_grayscale(img, kernel, 0)  # Pad với 0
    result = np.zeros_like(img, dtype=np.uint8)
    kh, kw = kernel.shape

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            region = padded_img[i:i+kh, j:j+kw]
            result[i, j] = np.clip(np.max(region + kernel), 0, 255)  # Giữ giá trị trong [0, 255]

    return result
```

source > morphological_operator > grayscale.py > grayscale_dilation

```python
def grayscale_erosion(img, kernel):
    """Grayscale Erosion: (f ⊖ b)(s,t) = min{f(s+x, t+y) - b(x,y)}"""
    padded_img = pad_image_grayscale(img, kernel, 255)  # Pad với 255 cho erosion
    result = np.zeros_like(img, dtype=np.uint8)
    kh, kw = kernel.shape

    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            region = padded_img[i:i+kh, j:j+kw]
            result[i, j] = np.clip(np.min(region - kernel), 0, 255)  # Giữ giá trị trong

    return result



def grayscale_opening(img, kernel):
    """Grayscale Opening: f ∘ b = (f ⊖ b) ⊕ b"""
    return grayscale_dilation(grayscale_erosion(img, kernel), kernel)

def grayscale_closing(img, kernel):
    """Grayscale Closing: f • b = (f ⊕ b) ⊖ b"""
    return grayscale_erosion(grayscale_dilation(img, kernel), kernel)
```

source > morphological_operator > grayscale.py > ...

```python
def grayscale_smoothing(img, kernel):
    """Grayscale Smoothing: Áp dụng Opening rồi Closing để làm mịn ảnh."""
    return grayscale_closing(grayscale_opening(img, kernel), kernel)

def grayscale_morphology_gradient(img, kernel):
    """Grayscale Morphology Gradient: (f ⊕ b) - (f ⊖ b)"""
    dilated = grayscale_dilation(img, kernel)
    eroded = grayscale_erosion(img, kernel)
    return np.clip(dilated.astype(np.int16) - eroded.astype(np.int16), 0, 255).astype(np.uint8)

def top_hat(img, kernel):
    """Top-hat Transformation: f - (f ∘ b)"""
    opened = grayscale_opening(img, kernel)
    return np.clip(img.astype(np.int16) - opened.astype(np.int16), 0, 255).astype(np.uint8)

def textural_segmentation(img, kernel):
    """Textural Segmentation: Dùng Top-hat để phân đoạn kết cấu"""
    return top_hat(img, kernel)
```

source > morphological_operator > grayscale.py > top_hat

```python
def granulometry(img, sizes):
    """Granulometry: Tính tổng giá trị pixel sau Opening với các kích thước kernel khác nhau"""
    granulometry_result = []
    for size in sizes:
        kernel = np.ones((size, size), dtype=np.uint8)
        opened = grayscale_opening(img, kernel)
        granulometry_result.append(np.sum(opened))
    return granulometry_result

def reconstruction(marker, mask, kernel, max_iter=100):
    """Morphological Reconstruction by Dilation: R_g^D(f)"""
    prev_marker = np.zeros_like(marker)
    iter_count = 0

    while not np.array_equal(marker, prev_marker):
        prev_marker = marker.copy()
        marker = np.minimum(grayscale_dilation(marker, kernel), mask)
        iter_count += 1
        if iter_count >= max_iter:
            print("Warning: Reconstruction reached max iterations!")
            break

    return np.clip(marker, 0, 255).astype(np.uint8)
```

source > morphological_operator > grayscale.py > reconstruction

```python

def reconstruction(marker, mask, kernel, max_iter=100):
    """Morphological Reconstruction by Dilation: R_g^D(f)"""
    prev_marker = np.zeros_like(marker)
    iter_count = 0

    while not np.array_equal(marker, prev_marker):
        prev_marker = marker.copy()
        marker = np.minimum(grayscale_dilation(marker, kernel), mask)
        iter_count += 1
        if iter_count >= max_iter:
            print("Warning: Reconstruction reached max iterations!")
            break

    return np.clip(marker, 0, 255).astype(np.uint8)

def create_structuring_element(size):
    """Tạo phần tử cấu trúc hình vuông"""
    return np.ones((size, size), dtype=np.uint8)

```

source > morphological_operator > binary.py > ...

```python
def convex_hull(img):
    """
    Tính toán bao lồi của một vùng sáng trong ảnh bằng thuật toán Jarvis March (Gift Wrapping).

    Parameters:
        img (numpy.ndarray): Ảnh nhị phân đầu vào (0: nền, 1: vùng sáng).

    Returns:
        numpy.ndarray: Ảnh nhị phân có đường bao lồi.
    """

    # Lấy danh sách tọa độ các điểm có giá trị 1 (điểm thuộc vùng sáng)
    points = np.argwhere(img == 1)

    # Nếu ảnh không có điểm sáng nào, trả về ảnh ban đầu
    if len(points) == 0:
        return img

    def cross_product(o, a, b):
        """
        Tính tích có hướng giữa hai vector oa và ob.
        Giá trị âm -> b nằm bên phải oa (ngược chiều kim đồng hồ).
        Giá trị dương -> b nằm bên trái oa (thuận chiều kim đồng hồ).
        """
        return (a[0] - o[0]) * (b[1] - o[1]) - (a[1] - o[1]) * (b[0] - o[0])

    # Tìm điểm có tọa độ (y, x) nhỏ nhất -> đây là điểm xuất phát của convex hull
    start = points[np.argmin(points[:, 1])]  # Chọn điểm có x nhỏ nhất (nếu trùng thì chọn y nhỏ nhất)
    hull = [start]  # Danh sách chứa các điểm của convex hull

    while True:
```

source > morphological_operator > binary.py > convex_hull > cross_product

```python
def convex_hull(img):

    while True:
        candidate = None  # Điểm kế tiếp trong convex hull
        for p in points:
            if np.array_equal(p, hull[-1]):  # Bỏ qua chính điểm hiện tại
                continue

            if candidate is None or cross_product(hull[-1], candidate, p) < 0:
                candidate = p  # Chọn điểm xa nhất theo chiều ngược kim đồng hồ

            elif cross_product(hull[-1], candidate, p) == 0:
                # Nếu ba điểm thẳng hàng, chọn điểm xa hơn
                if np.linalg.norm(p - hull[-1]) > np.linalg.norm(candidate - hull[-1]):
                    candidate = p

        if np.array_equal(candidate, start):  # Nếu quay lại điểm đầu tiên, thuật toán kết thúc
            break

        hull.append(candidate)  # Thêm điểm mới vào convex hull

    # Tạo ảnh nhị phân mới chứa bao lồi
    hull_img = np.zeros_like(img)
    for p in hull:
        hull_img[p[0], p[1]] = 1  # Đánh dấu các điểm thuộc đường bao lồi

    return hull_img
```

source > morphological_operator > binary.py > convex_hull

```python
def thinning(img):
    """Làm mỏng ảnh bằng thuật toán Zhang-Suen."""
    def count_transitions(P):
        """Đếm số lần chuyển từ 0 -> 1 theo thứ tự vòng."""
        P = [P[1,2], P[2,2], P[2,1], P[2,0], P[1,0], P[0,0], P[0,1], P[0,2], P[1,2]]
        return sum((P[i] == 0 and P[i+1] == 1) for i in range(8))

    def step(img, pass_num):
        markers = np.zeros_like(img)
        for i in range(1, img.shape[0] - 1):
            for j in range(1, img.shape[1] - 1):
                P = img[i-1:i+2, j-1:j+2]
                if img[i, j] == 1:
                    neighbors = np.sum(P) - 1
                    transitions = count_transitions(P)
                    cond1 = 2 <= neighbors <= 6
                    cond2 = transitions == 1
                    cond3 = P[0,1] * P[1,2] * P[2,1] == 0 if pass_num == 0 else P[1,2] * P[2,1] * P[1,0] == 0
                    if cond1 and cond2 and cond3:
                        markers[i, j] = 1
        img[markers == 1] = 0

    prev = np.zeros_like(img)
    while not np.array_equal(img, prev):
        prev = img.copy()
        step(img, 0)
        step(img, 1)

    return img
```

source > morphological_operator > binary.py > thinning

```python
def thickening(img, kernel):
    """Làm dày ảnh bằng dilation có điều kiện."""
    complement = np.logical_not(img)
    new_img = dilate(img, kernel)
    return np.logical_and(new_img, complement).astype(np.uint8)


def skeletonization(img):
    """Tạo bộ xương của đối tượng bằng phương pháp lặp erosion."""
    skeleton = np.zeros_like(img)
    temp = img.copy()
    while np.any(temp):
        eroded = erode(temp, np.ones((3, 3), np.uint8))
        skeleton = np.logical_or(skeleton, temp - eroded).astype(np.uint8)
        temp = eroded
    return skeleton


def reconstruction(marker, mask, kernel):
    """Tái tạo ảnh từ một marker bằng phương pháp giãn nở có giới hạn."""
    while True:
        next_marker = np.minimum(dilate(marker, kernel), mask)
        if np.array_equal(next_marker, marker):
            break
        marker = next_marker
    return marker
```

```python
def reconstruction(marker, mask, kernel):
    """Tái tạo ảnh từ một marker bằng phương pháp giãn nở có giới hạn."""
    while True:
        next_marker = np.minimum(dilate(marker, kernel), mask)
        if np.array_equal(next_marker, marker):
            break
        marker = next_marker
    return marker


def pruning(skeleton, iterations=1):
    """Cắt tỉa ảnh bộ xương bằng cách loại bỏ điểm cuối."""
    kernel = np.ones((3, 3), np.uint8)
    for _ in range(iterations):
        endpoints = np.logical_and(skeleton, np.sum(erode(skeleton, kernel), axis=(0, 1)) == 1)
        skeleton = np.logical_and(skeleton, np.logical_not(endpoints)).astype(np.uint8)
    return skeleton
```

main_grayscale.py

source > main_grayscale.py > apply_manual

```python
import sys
import getopt
import cv2
import numpy as np
import os
import time
import matplotlib.pyplot as plt
from morphological_operator.grayscale import  pad_image_grayscale, grayscale_dilation, grayscale_erosion, grayscale_oper


def apply_manual(img, kernel, seed=(10, 10)):
    """Áp dụng các phép toán hình thái grayscale bằng thuật toán tự viết."""
    print("apply_manual() function called")  # Kiểm tra xem hàm được gọi chưa

    print("Creating operations dictionary...")
    operations = {}
    sizes = [3, 5, 7]  # Các kích thước kernel cho Granulometry

    try:
        print("Adding Original...")
        operations["Original"] = img

        print("Adding Dilate...")
        operations["Dilate"] = grayscale_dilation(img, kernel)

        print("Adding Erode...")
        operations["Erode"] = grayscale_erosion(img, kernel)


        print("Adding Open...")
        operations["Open"] = grayscale_opening(img, kernel)
```

main_grayscale.py × | image_landscape.png | output_star_manual_all_grayscale.png | grayscale.py | output_landscape_manual_all_g

ource > main_grayscale.py > apply_manual

```python
def apply_manual(img, kernel, seed=(10, 10)):

        print("Adding Erode...")
        operations["Erode"] = grayscale_erosion(img, kernel)


        print("Adding Open...")
        operations["Open"] = grayscale_opening(img, kernel)

        print("Adding Close...")
        operations["Close"] = grayscale_closing(img, kernel)

        print("Adding Smoothing...")
        operations["Smoothing"] = grayscale_smoothing(img, kernel)

        print("Adding Gradient...")
        operations["Gradient"] = grayscale_morphology_gradient(img, kernel)

        print("Adding TopHat...")
        operations["TopHat"] = top_hat(img, kernel)

        print("Adding TexturalSegmentation...")
        operations["TexturalSegmentation"] = textural_segmentation(img, kernel)

        print("Adding Granulometry...")
        operations["Granulometry"] = granulometry(img, sizes)  # Trả về list, cần xử lý riêng khi hiển thị

        print("Adding Reconstruction...")
        operations["Reconstruction"] = reconstruction(img, img, kernel)  # Dùng img làm marker và mask

```

source > main_grayscale.py > apply_manual

```python
def apply_manual(img, kernel, seed=(10, 10)):
    except Exception as e:
        print(f"Error when adding to operations: {e}")
        sys.exit(1)

    print("Operations dictionary created:", list(operations.keys()))
    return operations

def apply_opencv(img, kernel, seed=(10, 10)):
    """Áp dụng các phép toán hình thái grayscale bằng OpenCV."""
    return {
        "Original": img,
        "Dilate": cv2.dilate(img, kernel, iterations=1),
        "Erode": cv2.erode(img, kernel, iterations=1),
        "Open": cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel),
        "Close": cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel),
        "Gradient": cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel),
        "TopHat": cv2.morphologyEx(img, cv2.MORPH_TOPHAT, kernel),
    }

def operator(in_file, out_file, mor_op, mode, wait_key_time=5000):
    """Thực hiện phép toán hình thái trên ảnh grayscale."""
    print(f"Đang xử lý file đầu vào: {in_file}")

    if not os.path.exists(in_file):
        print(f"Error: Input file '{in_file}' not found.")
        sys.exit(1)
```

source > main_grayscale.py > operator

```python
def operator(in_file, out_file, mor_op, mode, wait_key_time=5000):
    img = cv2.imread(in_file, cv2.IMREAD_GRAYSCALE)
    if img is None:
        print(f"Error: Unable to read image file '{in_file}'. Check file format and path.")
        sys.exit(1)
    print(f"Đã đọc ảnh: {img.shape}")

    # Chuẩn hóa ảnh về [0, 1] để phù hợp với tính toán
    img = img.astype(np.uint8)

    kernel = np.ones((3, 3), dtype=np.uint8)
    seed = (10, 10)

    start_time = time.time()

    if mode == "manual":
        operations = apply_manual(img, kernel, seed)
        method = "Manual (Custom)"
    elif mode == "opencv":
        operations = apply_opencv(img, kernel, seed)
        method = "OpenCV"
    else:
        print("Error: Invalid mode. Choose 'manual' or 'opencv'.")
        sys.exit(1)

    exec_time = time.time() - start_time
    mor_op = mor_op.capitalize() if mor_op else ""
    print(f"Danh sách phép toán khả dụng: {list(operations.keys())}")

```

source > main_grayscale.py > operator

```python
def operator(in_file, out_file, mor_op, mode, wait_key_time=5000):
    if mor_op:
        if mor_op in operations:
            result = operations[mor_op]
            if isinstance(result, list):  # Đặc biệt cho Granulometry
                print(f"Granulometry results for sizes [3, 5, 7]: {result}")
                # Không hiển thị ảnh cho Granulometry, chỉ lưu hoặc in kết quả
                np.savetxt(out_file.replace('.png', '.txt'), result, fmt='%.6f')
                print(f"Granulometry results saved to {out_file.replace('.png', '.txt')}")
            else:
                # Chuẩn hóa kết quả để hiển thị
                result_display = (result - result.min()) / (result.max() - result.min()) * 255
                result_display = result_display.astype(np.uint8)
                cv2.imshow(f"Result: {mor_op} - {method}", result_display)
                cv2.imwrite(out_file, result_display)
                cv2.waitKey(wait_key_time)
                cv2.destroyAllWindows()
                print(f"Output saved to {out_file}")
            print(f"Time Complexity ({method}): {exec_time:.6f} seconds")
        else:
            print(f"Error: Unknown morphological operation '{mor_op}'")
            print("Available operations:", list(operations.keys()))
            sys.exit(1)
```

source > main_grayscale.py > operator

```python
def operator(in_file, out_file, mor_op, mode, wait_key_time=5000):
    else:
        rows, cols = 2, 5  # Điều chỉnh lưới cho số lượng phép toán phù hợp
        images = [op for key, op in operations.items() if key != "Granulometry"]
        labels = [key for key in operations.keys() if key != "Granulometry"]

        h, w = images[0].shape
        label_height = 30
        grid_img = np.ones((h * rows + label_height * rows, w * cols), dtype=np.uint8) * 255

        for idx, (label, img) in enumerate(zip(labels, images)):
            if idx >= rows * cols:
                break
            row, col = divmod(idx, cols)
            y_start = row * (h + label_height)
            y_end = y_start + h
            x_start = col * w
            x_end = x_start + w

            if img.max() - img.min() == 0:
                img_display = np.zeros_like(img, dtype=np.uint8)  # Gán ảnh về 0 nếu không có sự thay đổi
            else:
                img_display = (img - img.min()) / (img.max() - img.min()) * 255
                img_display = img_display.astype(np.uint8)

            img_display = img_display.astype(np.uint8)
            grid_img[y_start:y_end, x_start:x_end] = img_display
            cv2.putText(grid_img, label, (x_start + 5, y_end + 20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, 0, 2)
```

source > main_grayscale.py > operator

```python
def operator(in_file, out_file, mor_op, mode, wait_key_time=5000):
        cv2.imshow(f"All Morphological Operations - {method}", grid_img)
        cv2.imwrite(out_file, grid_img)
        cv2.waitKey(wait_key_time)
        cv2.destroyAllWindows()
        print(f"All operations saved to {out_file}")
        import matplotlib.pyplot as plt

        if "Granulometry" in operations:
            granulometry_result = operations["Granulometry"]
            sizes = list(range(1, len(granulometry_result) + 1))  # Tạo danh sách kích thước SE

            # Vẽ biểu đồ Granulometry
            plt.figure(figsize=(8, 5))
            plt.plot(sizes, granulometry_result, marker='o', linestyle='-', color='b', label="Granulometry Profile")

            # Thêm tiêu đề và nhãn
            plt.xlabel("Structuring Element Size")
            plt.ylabel("Sum of Pixels After Opening")
            plt.title("Granulometry Analysis")
            plt.legend()
            plt.grid(True)

            # Lưu file hình ảnh
            img_filename = out_file.replace('.png', '_granulometry.png')
            plt.savefig(img_filename, dpi=300)  # Lưu với độ phân giải 300 dpi
            print(f"Granulometry plot saved to {img_filename}")

            # Hiển thị biểu đồ
            plt.show()
```

source > main_grayscale.py > operator

```python
def operator(in_file, out_file, mor_op, mode, wait_key_time=5000):
        print(f"Execution Time ({method}): {exec_time:.6f} seconds")


def main(argv):
    """Xử lý đầu vào từ dòng lệnh."""
    input_file = ''
    output_file = ''
    mor_op = ''
    mode = 'manual'
    wait_key_time = 5000

    description = 'Usage: main_grayscale.py -i <input_file> -o <output_file> [-p <morph_operator>] -m <mode> -t <wait_ke

    try:
        opts, args = getopt.getopt(argv, "hi:o:p:m:t:", ["in_file=", "out_file=", "mor_operator=", "mode=", "wait_key_t
    except getopt.GetoptError:
        print(description)
        sys.exit(2)

    for opt, arg in opts:
        if opt == '-h':
            print(description)
            sys.exit()
        elif opt in ("-i", "--in_file"):
            input_file = arg
        elif opt in ("-o", "--out_file"):
            output_file = arg
        elif opt in ("-p", "--mor_operator"):
            mor_op = arg
        elif opt in ("-m", "--mode"):
            mode = arg.lower()
```

source > main_grayscale.py > main

```python
def main(argv):
        elif opt in ("-i", "--in_file"):
            input_file = arg
        elif opt in ("-o", "--out_file"):
            output_file = arg
        elif opt in ("-p", "--mor_operator"):
            mor_op = arg
        elif opt in ("-m", "--mode"):
            mode = arg.lower()
        elif opt in ("-t", "--wait_key_time"):
            wait_key_time = int(arg)

    if not input_file or not output_file:
        print("Error: Missing required arguments.")
        print(description)
        sys.exit(1)

    operator(input_file, output_file, mor_op, mode, wait_key_time)

if __name__ == "__main__":
    main(sys.argv[1:])
```

III. Summarization of the usage

Detailed Usage and Algorithm Explanation:

GRAYSCALE IMAGE

1. Grayscale Dilation

• Definition

$$(f \oplus b)(s,t) = \max\{ f(s-x, t-y) + b(x,y) \mid (s-x), (t-y) \in D_f; (x,y) \in D_b \}$$

- **Usage**: Expands bright regions in a grayscale image by replacing each pixel with the maximum value in its neighborhood. Enhances bright structures.
- **Algorithm Explanation**:
    - Applies padding to the image to avoid boundary issues.
    - Iterates through the image, replacing each pixel with the maximum value within the structuring element.
    - Returns the dilated grayscale image.

2. Grayscale Erosion

• Definition

$$(f \Theta b)(s,t) = \min\{ f(s+x, t+y) - b(x,y) \mid (s+x), (t+y) \in D_f; (x,y) \in D_b \}$$

Usage: Shrinks bright regions by replacing each pixel with the minimum value in its neighborhood. Removes small bright details.

Algorithm Explanation:

- Applies padding to handle boundary effects.
- Iterates through the image, replacing each pixel with the minimum value within the structuring element.
- Returns the eroded grayscale image.

3. Grayscale Opening

• Definition

$$f \circ b = (f \Theta b) \oplus b$$

- **Usage**: Removes small bright spots by first performing erosion (shrinking objects) followed by dilation (restoring shape). Helps with noise reduction.
- **Algorithm Explanation**:
    - Applies grayscale erosion to shrink bright areas.
    - Applies grayscale dilation to restore larger structures.
    - Returns the processed grayscale image.

4. Grayscale Closing

• Definition

$$f \bullet b = (f \oplus b) \Theta b$$

Usage: Fills small dark gaps by first performing dilation (expanding objects) followed by erosion (shrinking back). Helps close small holes in bright areas.

Algorithm Explanation:

- Applies grayscale dilation to expand bright areas.
- Applies grayscale erosion to restore original object shapes.
- Returns the processed grayscale image.

5. Grayscale Smoothing

• Definition

$$h = (f \circ b) \bullet b$$

- **Usage**: Reduces noise and smooths object boundaries by applying both grayscale opening and closing sequentially.
- **Algorithm Explanation**:
  - First applies grayscale opening to remove small bright spots.
  - Then applies grayscale closing to fill small dark holes.
  - Returns the smoothed grayscale image.

6. Morphology Gradient

• Definition

$$h = (f \oplus b) - (f \Theta b)$$

- **Usage**: Highlights object edges by computing the difference between dilated and eroded versions of the image. Enhances edge features.
- **Algorithm Explanation**:
  - Computes the difference: **(Dilation - Erosion)**.
  - Bright areas indicate regions with

7. Top Hat

• Definition

$$h = f - (f \circ b)$$

- **Usage**: Extracts small bright details by subtracting the opened image from the original. Useful for enhancing fine structures.
- **Algorithm Explanation**:
  - Applies **grayscale opening** to remove small bright regions.
  - Subtracts the opened image from the original image, preserving only small bright details.
  - Returns the top-hat transformed image.

8. Textual segmentation

• Definition

$$h = (f \bullet b_1) \circ b_2$$

- **Usage**: Enhances texture features in an image by extracting fine structures using the top-hat transformation.
- **Algorithm Explanation**:
  - Uses **top-hat transformation** to highlight small bright details.

- Enhances textural patterns by emphasizing variations in local brightness.
- Returns the segmented image emphasizing textures.

9. Granulometry

- **Usage**: Analyzes the size distribution of bright structures by applying **grayscale opening** with different structuring element sizes.
- **Algorithm Explanation**:
  - Iterates through different kernel sizes.
  - Applies **grayscale opening** at each size to filter out smaller bright regions.
  - Computes the sum of pixels after each opening, indicating the presence of structures of different sizes.
  - Returns a list of summed pixel values for different kernel sizes, used to plot a granulometry curve.

10. Reconstruction

1. Opening by reconstruction of the original image using a horizontal line of size 1x71 pixels in the erosion operation

$$O_R^{(n)}(f) = R_f^D[f \ominus nb]$$

2. Subtract the opening by reconstruction from original image

$$f' = f - O_R^{(n)}(f)$$

3. Opening by reconstruction of the f' using a vertical line of size 11x1 pixels

$$f1 = O_R^{(n)}(f') = R_f^D[f' \ominus nb']$$

4. Dilate f1 with a line SE of size 1x21, get f2.

5. Calculate the minimum between the dilated image f2 and and f', get f3.

6. By using f3 as a marker and the dilated image f2 as the mask,

$$R_{f2}^D(f3) = D_{f2}^{(k)}(f3)$$

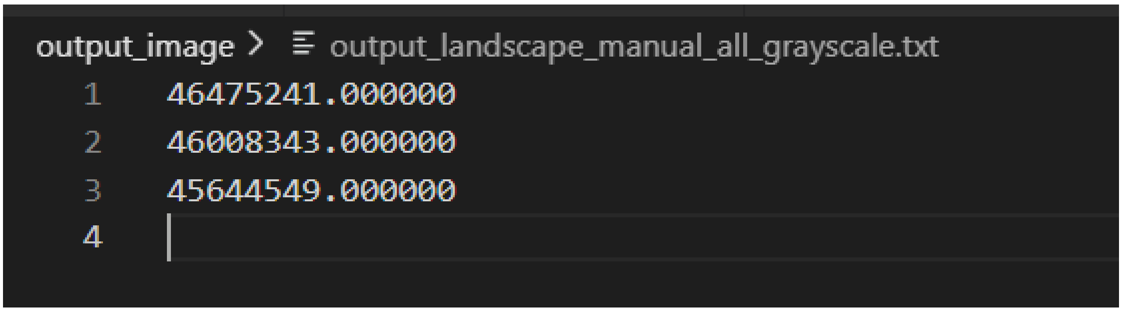
$$\text{with k such that } D_{f2}^{(k)}(f3) = D_{f2}^{(k+1)}(f3)$$

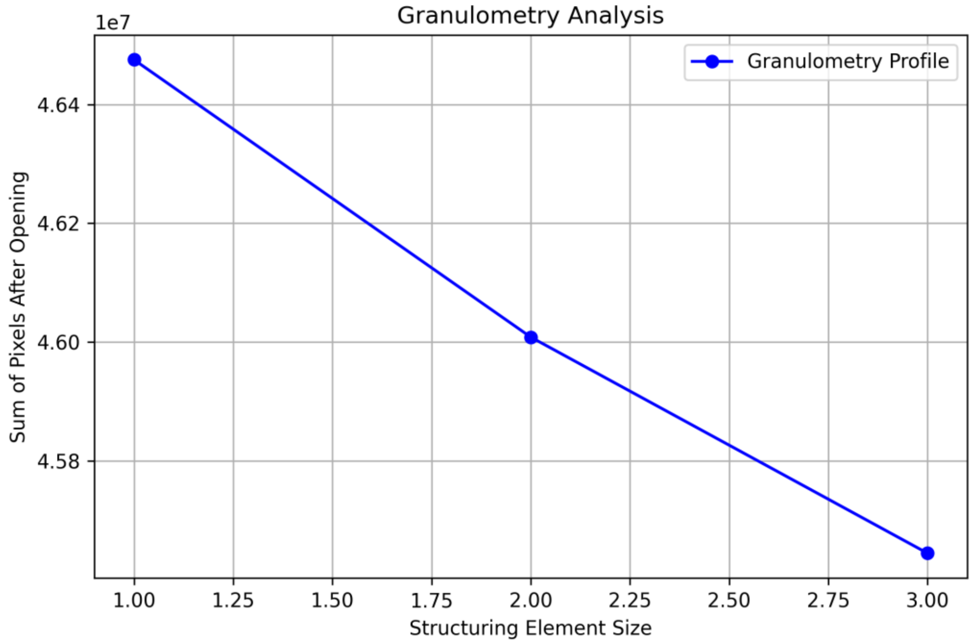
- **Usage**: Restores missing or occluded image regions by iteratively applying **dilation** constrained by a mask.
- **Algorithm Explanation**:
  - Initializes a **marker** image, typically a subset of the mask.
  - Iteratively dilates the marker while ensuring it does not exceed the mask.
  - Stops when the marker no longer changes between iterations or reaches **max_iter**.
  - Returns the reconstructed grayscale image.

IV. EXPERIMENTS AND EVALUATION:

Grayscale image.

The results of granulometry function on landscape grayscale image





The Granulometry output file now contain different values

granulometry function is now producing **three distinct values:**

- **46475241.000000**
- **46008343.000000**
- **45644549.000000**

What do these values represent?

Each value corresponds to the **sum of pixel intensities** after applying **Grayscale Opening** with different structuring element (SE) sizes.

- The first value (**46475241.000000**) corresponds to **SE size 3x3**.
- The second value (**46008343.000000**) corresponds to **SE size 5x5**.
- The third value (**45644549.000000**) corresponds to **SE size 7x7**.

Why do the values decrease?

As the structuring element **size increases**, **more small bright regions are removed**, leading to a **reduction in total pixel intensity**.

- **Smaller SEs (3x3)** only remove very small noise.

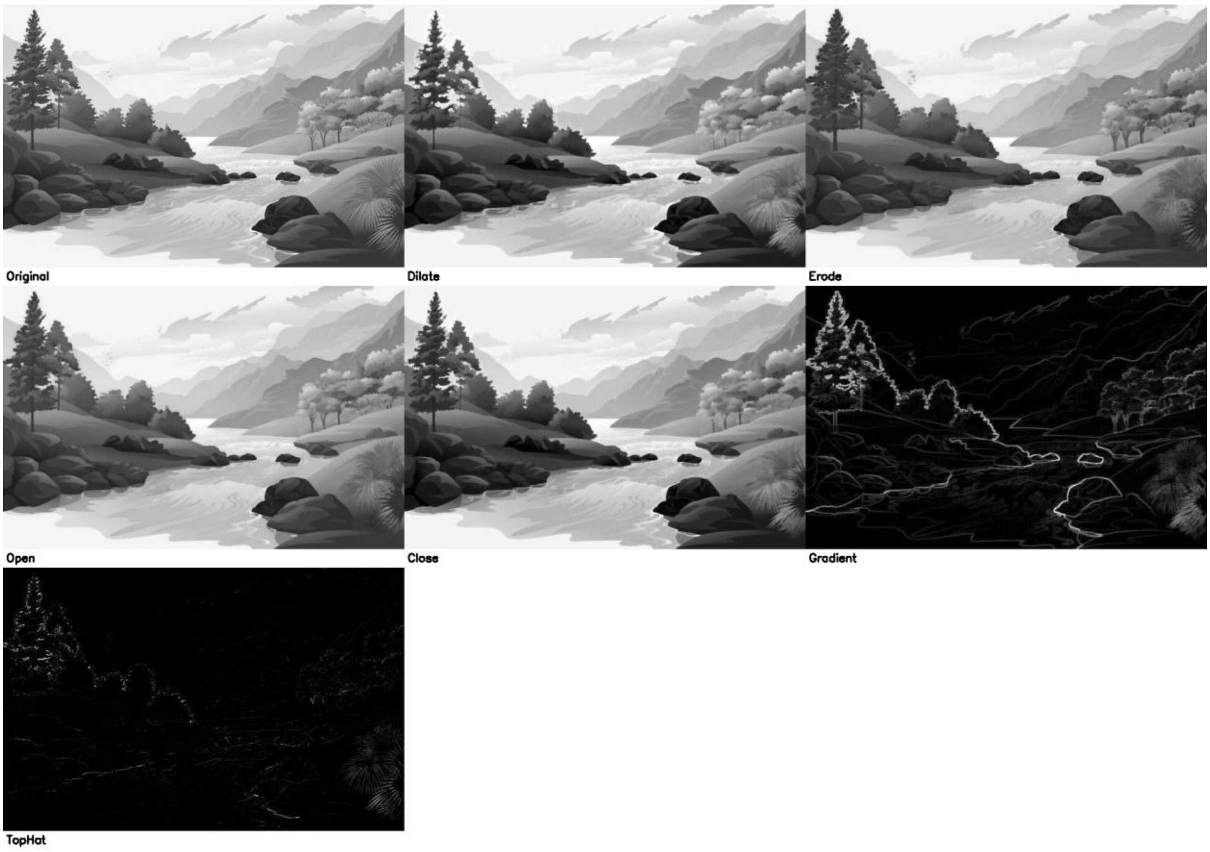
- **Larger SEs (7x7)** remove larger details and thin structures, reducing the total pixel sum.

What does this tell us about the image?

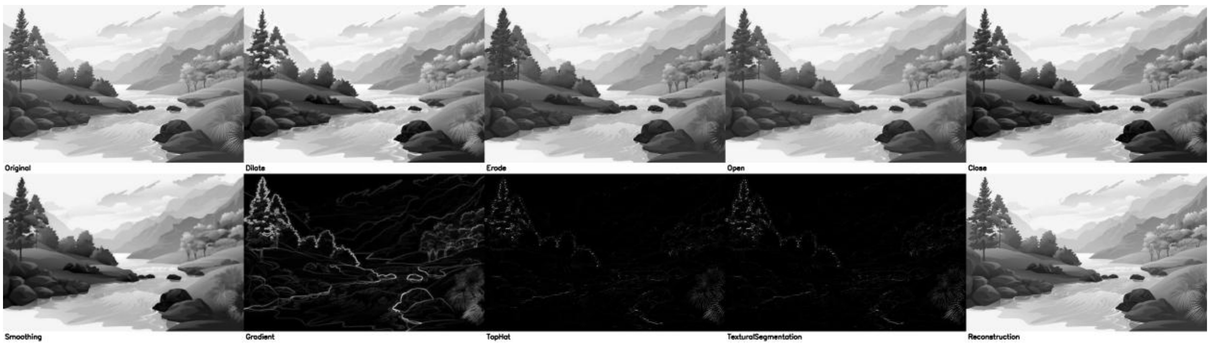
- The image **contains small bright structures** that are gradually removed as SE size increases.
- The decrease in values suggests **progressive removal of finer details**, which aligns with granulometry analysis.
- If values stayed **constant**, it would mean **no small structures were removed**, or SE sizes were **too small** to affect the image.

<u>**The results of other custom grayscale morphological operations on landscape grayscale image**</u>

Opencv results



Manual results:



Comments:

The custom grayscale morphological operations produce results that often differ from OpenCV's built-in functions, with some outputs appearing darker or less refined. OpenCV generally provides smoother and more accurate transformations due to optimized padding, normalization, and efficient kernel application. The main issues in the custom implementation likely stem from **inefficient pixel scaling, boundary handling, and loop-based computations**. Refining these aspects could improve accuracy and bring results closer to OpenCV’s optimized output.

Comparison in general

Grayscale image operator custom vs opencv built in

Aspect	Custom Implementation	OpenCV Implementation
Performance (Speed)	Slow (nested loops, sequential processing)	Fast (optimized C++ backend, SIMD, parallel processing)
Accuracy	Can be prone to errors (padding issues, structuring element misalignment)	Highly reliable and precise
Dilation & Erosion	Works if implemented correctly but slower	Optimized and fast
Opening & Closing	Affected by dilation/erosion accuracy	Standardized and efficient
Hit-or-Miss	May introduce errors if background processing is incorrect	Handles structuring elements correctly
Boundary Extraction/ Gradient	Accurate if erosion is properly done	Faster with correct results
Region Filling	Uses iterative dilation, slow for complex shapes	Uses flood fill, faster and more memory-efficient
Thinning	Works but may leave artifacts if conditions aren't checked properly	Optimized and standardizes Zhang-Suen or Guo-Hall algorithms