Лабораторна робота № 9

# ВИЗНАЧЕННЯ ДЕФЕКТИНХ ЗОН ЗА ДОПОМОГОЮ РОЗРАХУНКУ ФРАКТАЛЬНОЇ РОЗМІРНОСТІ

## 9.1 Мета роботи

Дослідити залежність впливу дефектних зон (тріщини/вкраплення/інклюзії) на значення фрактальної розмірності.

Дослідити покращення алгоритму adaptive box-counting інтегральними зображеннями.

## 9.2 Основні теоретичні відомості

Фрактал — це крива або математична структура з певним повторюваним візерунком, який може бути конгруентним або неконгруентним. Вони самоподібні в різних масштабах. Вони створюються шляхом рекурсивного повторення простого процесу знову і знову. Фрактали можна інтерпретувати як зображення, що представляють «Порядок у хаосі». На перший погляд вони здаються дуже хаотичними та містять складність, що перевищує розуміння. Але процес, у якому створюються фрактали, є дуже впорядкованим і може бути визначений законами математики. Фрактали можуть бути симетричними та асиметричними. Єдиний спосіб зрозуміти та проаналізувати таку геометричну фігуру – це кількісно визначити її складність, що, по суті, і є фрактальною розмірністю. Це міра складності фігури. Чим вища складність, тим вища фрактальна розмірність.

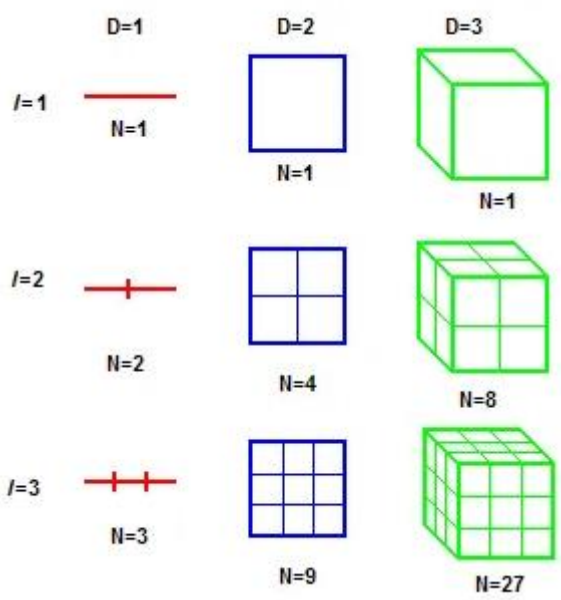


Рис.9.1 – Візуалізація фрактальної розмірності.

1. Як показано на зображенні вище, пряма лінія має лише один вимір. Якщо взяти коефіцієнт множення (R) та вимір (D) і помножити довжину лінії на 2 (тобто R=2), ми отримаємо фактично дві лінії. Припустимо, що кількість виходів дорівнює N. Отже, для R=2 та D=1 ми отримуємо N=2.

2. У другому вимірі (тобто D=2) ми можемо мати квадрат зі стороною L. Але якщо помножити довжину прямої на 2 (тобто R=2), ми отримаємо чотири квадрати (тобто $R^2$). Для R=3 ми отримаємо дев'ять квадратів (тобто $R^2$) і так далі.

3. Для 3-го виміру (D=3) ми можемо мати куб зі стороною L. Помноживши пряму на 2 (R=2), ми отримаємо вісім кубів. Для R=3, D=3, ми отримаємо N=27 (тобто $R^3$)

Спостерігаючи за вищезазначеними пунктами, стає чітко очевидним, що для правильних симетричних фігур

$$D = \frac{\log N}{\log R}, \tag{9.1}$$

Для кривої Коха, як показано на зображенні нижче, важливо врахувати, що для кожного нового порядку існують чотири повторювані фігури з попереднього порядку. Наприклад, фігура в порядку =1 має чотири прямі лінії від фігури в порядку =0. Аналогічно, фігура в порядку =2 – це, по суті, фігура з порядку =3, яка повторюється чотири рази. Отже, можна сказати, що R=3 та N=4 для порядку =3. Підставляючи ці значення у формулу, отримуємо D=1,262 для порядку =3, що є фрактальним виміром фігури.

Box-counting – це метод збору даних для аналізу складних візерунків шляхом розбиття набору даних, об'єкта, зображення тощо на все менші й менші частини, зазвичай у формі "коробки", та аналізу цих частин у кожному меншому масштабі. Суть цього процесу порівнюють зі збільшенням або зменшенням масштабу за допомогою оптичних або комп'ютерних методів для дослідження того, як спостереження деталей змінюються з масштабом. Однак, під час підрахунку коробок, замість зміни збільшення або роздільної здатності лінзи, дослідник змінює розмір елемента, який використовується для огляду об'єкта або візерунка. Цей метод зазвичай реалізується в програмному забезпеченні для використання на візерунках, витягнутих з цифрових носіїв, хоча фундаментальний метод може бути використаний для фізичного дослідження деяких візерунків. Метод виник і використовується в фрактальному аналізі . Він також має застосування в суміжних галузях, таких як лакунарність та мультифрактальний аналіз.

Відповідні ознаки, зібрані під час підрахунку коробок, залежать від досліджуваного об'єкта та типу проведеного аналізу. Наприклад, двома добре вивченими об'єктами підрахунку коробок є бінарні (тобто такі, що мають лише два кольори, зазвичай чорний і білий) та сіро-градаційні цифрові зображення (тобто jpeg, tiff тощо). Підрахунок коробок зазвичай проводиться на основі шаблонів, отриманих з таких нерухомих зображень, і в цьому випадку записана необроблена інформація зазвичай базується на характеристиках пікселів, таких як заздалегідь визначене значення кольору або діапазон кольорів чи інтенсивності. Коли підрахунок коробок проводиться для визначення фрактальної розмірності , відомої як розмірність підрахунку коробок , записана інформація зазвичай є або так, або ні, щодо того, чи містить коробочка будь-які пікселі заздалегідь визначеного кольору або діапазону тобто кількість коробок, що містять відповідні пікселі в кожній підраховується). Для інших типів аналізу шуканими даними можуть бути кількість пікселів, що потрапляють у вимірювальну рамку, діапазон або середні значення кольорів чи інтенсивності, просторове розташування пікселів у кожній рамці або такі властивості, як середня швидкість (наприклад, від потоку частинок).

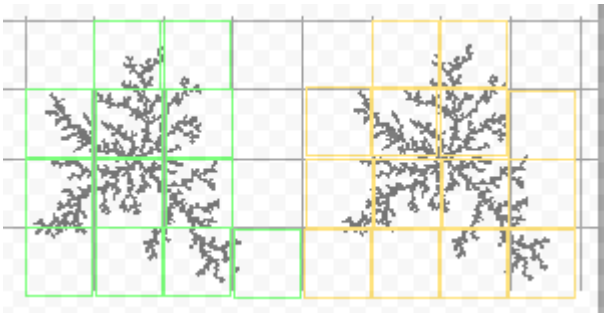

Рис.9.2 Покриття фрактального об’єкта множини з коробок.

Бінаризація зображення, також відома як встановлення порогу зображення, – це техніка створення бінарного зображення з зображення у градаціях сірого або RGB, яку можна використовувати для відокремлення переднього плану зображення від фону. Встановлення порогу зображення – один із найфундаментальніших способів вилучення корисної інформації з заданого зображення або сегмента області інтересу.

Порогове значення зображення зазвичай виконується для зображень у градаціях сірого. Після перетворення зображення на зображення у градаціях сірого кожне значення пікселя порівнюється з пороговим значенням, де, якщо значення пікселя менше за порогове значення пікселя встановлюється на нуль; в іншому випадку воно встановлюється на максимально можливе значення (255).

У глобальному пороговому режимі до кожного пікселя застосовується фіксоване порогове значення, припускаючи, що зображення має бімодальну гістограму, яка відділяє фон від переднього плану. Алгоритм глобального порогового регулювання порівнює інтенсивність кожного пікселя у вихідному зображенні з попередньо визначеним пороговим значенням і перепризначає різні значення інтенсивності, якщо інтенсивність більша або менша за порогове значення.

Оскільки порогове значення фіксоване в глобальному пороговому управлінні, воно погано працює в умовах змінного освітлення та тіні. Адаптивне порогове управління, локальний метод порогового управління, допомагає подолати вищезгаданий недолік, обчислюючи різні значення порогового управління для різних областей зображення.

У OpenCV цю cv2.adaptiveThreshold функцію можна використовувати для виконання адаптивного порогового значення для заданого зображення. OpenCV пропонує два алгоритми/методи адаптивного порогового значення, які обчислюють значення порогу за допомогою різних механізмів.

Область інтересу (часто скорочено ROI ) – це зразок у наборі даних, визначеному для певної мети. Концепція ROI зазвичай використовується в багатьох сферах застосування. Вона існує як околиця або в межах однієї області. Наприклад, у медичній візуалізації межі пухлини можуть бути визначені на зображенні або в об'ємі з метою вимірювання її розміру. Ендокардіальна межа може бути визначена на зображенні, можливо, під час різних фаз серцевого циклу, наприклад, в кінці систоли та в кінці діастоли, з метою оцінки функції серця. У географічних інформаційних системах (ГІС) ROI можна розуміти буквально як полігональне виділення з 2D-карти. У комп'ютерному зорі та оптичному розпізнаванні символів ROI визначає межі об'єкта, що розглядається. У багатьох застосуваннях до ROI додаються символічні (текстові) мітки, щоб описати її вміст у компактній формі. У межах ROI можуть знаходитися окремі точки інтересу (POI).

## 9.3 Підготовка до роботи

Проаналізувати метод для визначення фрактальної розмірності box-counting визначити слабкі та сильні сторони алгоритму. Проаналізувати код в додатку 1.

Проаналізувати методи для визначення фрактальної розмірності адаптивного поділу box-counting з інтегральними зображеннями визначити переваги та недоліки. Проаналізувати код в додатку 2.

## 9.4 Виконання роботи

1. Завантажити в оперативну пам’ять початкове зображення **поверхні** бурштину задане викладачем;

2. Провести обчислення фрактальної розмірності методом box-counting та записати значення та швидкість виконання для зображення;

3. Провести обчислення фрактальної розмірності методом адаптивного поділу box-counting з інтегральними зображеннями та записати значення та швидкість виконання для зображення;

4. Повторити п.2 і п.3 для ще 2х зображень поверхні бурштину заданих викладачем;

5. Завантажити в оперативну пам’ять **зображення** бурштину заданого викладачем та виконати:

- виокремлення ROI
- формування ядра
- провести пошук оптимального розміру ядра
- провести пошук зсуву ядра для визначення дефектних зон (знайдені дефект мають відображатись на зображені червоним кольором).

## 9.5 Зміст звіту

1. Найменування і мета роботи;

2.Графік значення фрактальної розмірності для 2х алгоритмів по 3х зображень поверхні бурштину;

3. Графік порівняння швидкості роботи для 2х алгоритмів по 3м зображень поверхні бурштину;

4. Зображення з різним розміром ядра та кроком підібраним власноруч.

## 9.6 Контрольні запитання

1. зображень поверхні бурштину

2. Яка основна проблема глобальної бінаризації фіксованим порогом?

3. Опишіть, як вибір вікна для адаптивної бінаризації впливає на тонкі структури та загальний контраст.

4. Що найбільше спотворює оцінку D на дрібних масштабах?

5. Де найчастіше застосовуються інтегральні зображення?

```
def fractal_dimension(Z: np.ndarray, threshold=0.9) -> np.float64:

    assert(len(Z.shape) == 2)
    def boxcount(Z, k):
        S = np.add.reduceat(
            np.add.reduceat(Z, np.arange(0, Z.shape[0], k), axis=0),
                            np.arange(0, Z.shape[1], k), axis=1)
        return len(np.where((S > 0) & (S < k*k))[0])

    Z = (Z < threshold)

    p = min(Z.shape)

    n = 2**np.floor(np.log(p)/np.log(2))

    n = int(np.log(n)/np.log(2))
      sizes = 2**np.arange(n, 1, -1)

    counts = []
    for size in sizes:
        counts.append(boxcount(Z, size))

    coeffs = np.polyfit(np.log(sizes), np.log(counts), 1)
    return -coeffs[0]
```

```
def binarize(image, threshold=0.9):
    """Перетворення зображення в бінарне (True/False)"""
    return image > threshold

def compute_integral_image(binary_image):
    """
    Обчислює інтегральне зображення для швидкого підрахунку суми пікселів у підрегіонах.
    Інтегральне зображення має розмір (h+1, w+1) з нульовим заповненням на початку.
    """
    return np.pad(np.cumsum(np.cumsum(binary_image.astype(np.int32), axis=0), axis=1),
                  ((1, 0), (1, 0)), mode='constant', constant_values=0)

def subdivide(cell):
    """Поділ клітинки на 4 підклітинки"""
    x, y, w, h = cell
    half_w, half_h = w // 2, h // 2
    return [
        (x, y, half_w, half_h),
        (x + half_w, y, half_w, half_h),
        (x, y + half_h, half_w, half_h),
        (x + half_w, y + half_h, half_w, half_h)
    ]

def linear_regression_slope(x, y):
    """Обчислення нахилу прямої за методом найменших квадратів"""
    coeffs = np.polyfit(x, y, 1)
    return coeffs[0]

def fractal_dimension(image, min_cell_size=2, threshold=0.9):
    """
    Обчислює фрактальну розмірність зображення за допомогою адаптивного алгоритму,
    який використовує інтегральне зображення для швидкого підрахунку клітинок.

    Відрізняємо клітинки за статусом:
      - "empty": повністю порожня (не належить фракталу)
      - "full": повністю заповнена (належить фракталу)
      - "mixed": містить як фрактальні, так і порожні пікселі

    При підрахунку враховуємо як "full", так і "mixed" клітинки.
    """
    binary_image = binarize(image, threshold)
    height, width = binary_image.shape
    init_size = min(width, height)


    int_img = compute_integral_image(binary_image)

    def cell_status(cell):
        """Повертає статус клітинки: 'empty', 'full' або 'mixed'"""
        x, y, w, h = cell
        total = int_img[y + h, x + w] - int_img[y, x + w] - int_img[y + h, x] + int_img[y, x]
        if total == 0:
            return "empty"
        elif total == w * h:
            return "full"
        else:
            return "mixed"

    active_cells = [(0, 0, init_size, init_size)]
    scale_data = []
    current_cell_size = init_size

    while current_cell_size >= min_cell_size:
```

```
    count_active = 0
    new_active_cells = []
    for cell in active_cells:
        status = cell_status(cell)

        if status in ("full", "mixed"):
            count_active += 1
        if status == "mixed" and current_cell_size > min_cell_size:
            new_active_cells.extend(subdivide(cell))
    scale_data.append((current_cell_size, count_active))
    active_cells = new_active_cells
    current_cell_size = current_cell_size // 2

epsilons = np.array([sd[0] for sd in scale_data])
counts = np.array([sd[1] for sd in scale_data])
log_eps = np.log(epsilons)
log_counts = np.log(counts + 1e-8)

slope = linear_regression_slope(log_eps, log_counts)
fractal_dim = -slope
return fractal_dim
```