

2. Алгоритм проходження контуром об'єкта з використанням зворотного ходу

В багатьох задачах обробки зображень аналіз форм об'єктів є актуальним і інформативним. При аналізі біомедичних зображень важливу роль відіграє дослідження форми мікрооб'єктів. Оскільки основна інформація про форму об'єкта міститься в контурі об'єкта, то виділення та опис контуру є важливою задачею аналізу зображень. Для визначення контуру використовують такі підходи: підкреслення різких перепадів яскравості або виділення однорідних областей з подальшим проходженням контуром. Другий підхід ми і будемо використовувати в цьому дослідженні.

Серед відомих алгоритмів виділення областей є такі: порогова сегментація, кластеризація, нарощування областей, алгоритм водоподілу, блочна сегментація тощо. Дані алгоритми базуються на об'єднанні пікселів в однорідні області на основі певного закону однорідності або ознаки. Результатом їх роботи є набір однорідних областей. Для отримання зв'язного контуру об'єкта необхідно використати алгоритми проходження контуром.

Проходження контуром – процес послідовного перебору пікселів цифрового зображення за певними правилами з метою знаходження зв'язного контуру об'єкта. Метою нашого дослідження, якраз і буде аналіз цих зображень.

Визначення, які необхідні для розуміння алгоритму:

Стартовий піксел P_s – піксел, з якого починається обхід контуром об'єкта. Вибір стартової точки проводиться довільно, наприклад крайній лівий верхній піксел, що належить об'єкту.

Кінцевий піксел P_e – піксел, потрапляючи на який алгоритм завершує свою роботу.

Активний піксел P_a – піксел, що знаходиться в середині розмітаної сітки.

Сусідній піксел P_n – піксел, який межує з активним пікселем.

Контурний піксел P_c – піксел, що належить контуру об'єкта.

Фоновий піксел P_f – піксел, що належить фону зображення.

Сусідній контурний піксел P_{cn} – піксел, що належить контуру об'єкта та межує з активним пікселем.

Покроковий опис алгоритму «Backward contour tracing»

- **1 крок.**
Проводимо пошук стартового пікселя P_a , пройшовшись по списку усіх пікселів додаємо його до списку пікселів контуру.
- **2 крок.**
Проводимо пошук сусіднього контурного пікселя P_n (піксель, який є тим самим кольором що і наш стартовий піксель або з дуже невеликою різницею в кольорі) проходячи по восьми пікселях за годинниковою стрілкою починаючи від верхнього лівого.
- **3 крок.**
Проводимо пошук сусіднього контурного пікселя P_n проти годинникової стрілки.
- **4 крок.**
Якщо отримані сусідні контурні пікселі в кроці 2 та 3 співпадають, $P_n == P_n$ то активний піксель (той відносно якого здійснюється пошук) визнається як фоновий P_f і виключається з подальшої обробки (тобто перестає бути контуром) і здійснюється перехід до 8 кроку.
- **5 крок.**
Якщо отримані сусідні контурні пікселі не співпадають $P_n != P_n$, то стартовий піксель визнається також і кінцевим $P_s = P_e$. Сусідній контурний піксель, отриманий на кроці 2, заноситься в масив контурних пікселів і йому присвоюється мітка активного пікселя $P_a = P_{cn}$.
- **6 крок.**
Проводиться пошук наступного сусіднього контурного пікселя P_{cn} . Послідовність перевірок сусідніх контурних точок відбувається на основі розмітаної сітки. Піксели перевіряються за годинниковою стрілкою. Позиція стартової перевірки d визначається як $(d'+2) \bmod 8$, де d' – позиція, з якої було знайдено активний піксель P_a . Пошук завершується при знаходженні наступного контурного пікселя або до перевірки всіх сусідніх контурних пікселів.
- **7 крок.**
Якщо сусідній піксель є контурним та не співпадає з кінцевим пікселем, то він заноситься в масив контурних пікселів і йому присвоюється мітка активного пікселя $P_a = P_{cn}$ та проходить перехід назад до кроку 6.
- **8 крок.**
Якщо знайдений сусідній піксель розмічений на попередніх кроках пошуку, але не співпадає з кінцевим пікселем $P_{cn} != P_e$, то активний піксель визнається малоінформативним, видаляється з масиву контурних точок, індексується як піксель фону. Статус активного пікселя присвоюється попередньому контурному пікселю.
- **9 крок.**
Якщо знайдений контурний піксель співпадає з кінцевим пікселем $P_{cn} == P_e$ та кількість точок, що належать контуру, більша за 1, то алгоритм завершує роботу.

Псевдокод

```
def cont(Ps, all_pix):
    conts = []
    pos = Ps.position#position of Ps in all_pix
    Pa = Ps
    conts.append(Pa)
    d = 0
    for i in range 8:
        Pn1 = all_pix[pos + i]#Pn1 are pixel next to Pa
        if Pn1.c:#if Pn is contouring pixel
            Pcn = Pn1
            break
    for j in range(7, -1, -1):
        Pn2 = all_pix[pos + i]#Pn2 are pixels next to Pa
        if Pn2.c:
            Pcn2 = Pn2
            break
    if Pcn == Pcn2:
        Pa.c = false
        broken = true
    else:
        Pe = Pa
        d = i
        Pa = Pcn
        conts.append(Pa)
        while True:
            for i in range 8:
                Pn = all_pix[Pa.position + d + i]#starting from d we check pixels
                if Pn.c and Pn != Pe and Pn not in conts broken = false:
                    d = (d + 6) % 8
                    Pa = Pn
                    conts.append(Pa)
            else if (Pn.c and Pn != Pe) or (broken = true and Pn.c and Pn != Pe):
                Pa.c = false#this pixel becomes background
                conts.remove(Pa)
                if broken = true:
                    Pa = Pcn
                    d = i
                else:
                    Pa = conts[-1]
            else if Pn == Pe and len(conts) > 1:
                return conts
```

Складність алгоритму

Спершу щоб знайти стартову точку ми мусимо пройтись по масиву всіх пікселів, щоб знайти нам потрібний що має складність $O(n)$ проте можливо нам вдасться реалізувати цей пошук за $O(\log(n))$. Якщо позначити кількість усіх контурних пікселів змінною k то час проходження по кожному з них буде $O(8*k) = O(k)$ тобто загальний час вирішення задачі буде $O(n + k)$.

Опис експерименту

Отож, для експерименту брались зображення різної важкості, задля тестування нашого алгоритму «проходження контуром об'єкта з використанням зворотного ходу». Усі картинки є в папці Images у Github репозиторії. Для прикладу зрівнювали його з Theo Pavlidis algorithm:

Алгоритм	Периметр, точки	Кількість перевірок	Час роботи
Theo Pavlidis algorithm	1283	2683	5.07
«Backward contour tracing»	1283	2311	3.95

Ще було додано інтерфейс для зручності роботи, а саме бібліотеки cv2(зчитує точку яку ви вибрали на зображенні, її копію), tkinter.filedialog(за допомогою цієї бібліотеки ви просто вибираєте зображення з папки, вказуєте точку з якої хочете починати відлік на зображенні).

На відео ми демонструємо, як працює наш алгоритм(посилання, також, в репозиторії):

Ми вибираємо картинку, яку хочемо дослідити -> вказуємо точку -> спершу виводиться стартове зображення з обведеним контуром -> а наступне наш результат в виді контуру об'єкту.

Висновки

- Після виконання цього дослідження ми навчились працювати із зображеннями та знаходити контур об'єктів, які ми потребуємо з цих прикладів. Зробили висновок, що цей алгоритм можна і надалі модифікувати для конкретних задач.
- Найбільшу складність при виконанні цього завдання склали ті зображення, які мають дуже незначний перехід між кольорами і було важко розбити їх на окремі сегменти задля знаходження контуру конкретного об'єкту.
- Якраз вирішення цієї проблеми на даний момент лишилось незрозумілим, як розбити зображення на сегменти з мінімальною різницею кольорів. Повністю цієї проблеми нам так і не вдалось уникнути.