Segmentacja wododzialowa

autorzy: Krystian Molenda, Anna Cięciara

O projekcie

> Tematem projektu jest implementacja algorytmu segmentacji wododziałowej.
> Segmentacja wododziałowa polega na podziale obrazu na regiony, będące zbiorami pikseli. Regiony są jednorodne pod względem pewnych własności.

> Segmentacja wododziałowa polega na wyznaczaniu linii rozdzielających potencjalne obszary zlewiskowe. Proces segmentacji symuluje zalewanie coraz wyżej położonych obszarów. Segmentację rozpoczynamy od pikseli najjaśniejszych a kończymy na najciemniejszych.

> Aby rezultat segmentacji był satysfakcjonujący należy przygotować odpowiedni gradient obrazu, na którym wykonywany będzie algorytm. Istnieje wiele możliwośi utworzenia gradientu obrazu. Zależnie od wybranej, efekt może być inny.

> W projekcie zajmuję się implementacją algorytmu segmentacji wododziałowej Vincent-Soille. Podczas pracy opieraliśmy się na następującej publikacji

> Aby rezultat segmentacji był satysfakcjonujący należy przygotować gradient obrazu, na którym wykonywany będzie algorytm. Istnieje wiele możliwośi utworzenia gradientu obrazu. Zależnie od wybranej metody efekt może być inny.

Algorytm

> Program został oparty na algorytmie **Vincent-Soille Waterhsed - Watershed by Immersion** - Algorithm 4.1 z publikacji

Prezentacja algorytmu

```
1: procedure: Watershed-by-Immersion
2: Input: Gradient obrazu G = (D, E, im).
3: Output: Macierz etykiet lab.

4: define init – 1              # wartość inicjalizacyjna macierzy etykiet
5: define mask – 2              # wartość początkowa zbiornika
6: define wshed 0               # etykieta pikseli wododzialowych
7: define fictitious (–1, –1)   # znacznik - piksel fikcyjny o nierealnych
współrzędnych
8: curlab←0                     # aktualna warstwa
9: fifo init(queue)             # kolejka FIFO

    # inicjalizacja macierzy na podstawie pikseli z oryginalnego obrazu
10: for all p ∈ D do            # Iteracja po każdym pikselu obrazu wejściowego
11:     lab[p]←init             # Inicjalizacja macierzy etykiet
12:     dist[p]←0               # Inicjalizacja macierzy odległości
```

```
13: end for
14: SORT piksele po poziomach szarości
```

```
        # Początek zalewania
15: for h = hmin to hmax do #Iteracja po wszystkich odcieniach szarości od
najniższego
16:     for all p ∈ D with im[p] = h do # Iteracja po pikselach w danym odcieniu
szarości
17:         lab[p] ←mask # Ustawiamy wartość początkową w macierzy etykiet dla
danego                              piksela

            # Jeśli dla piksela p istnieje sąsiad q który: jest zbiornikiem lub
jest pikselem wododzialowym
18:         if p has a neighbour q with (lab[q] > 0 or lab[q] = wshed) then
19:             dist[p]←1           # Oznaczamy piksel p
20:             fifo add(p, queue)  # Dodajemy piksel do kolejki
21:         end if
22:     end for  # Dla każdego poziomu szarości będziemy otrzymywać piksele,
posiadające sąsiadów z etykietą
```

```
23:     curdist←1 # Deklarujemy zasięg zalewania
24:     fifo add(fictitious, queue) # Dodajemy do kolejki znacznik

25:     while True # Pętla rozszerzająca zalany obszar
26:         p ←fifo remove(queue) # pobieramy pierwszy dodany do kolejki piksel
(first
in)
27:         if p = fictitious then # Jesli fikcyjny
28:             if fifo empty(queue) then # Jeśli kolejka pusta przerywamy pętlę
29:                 break
30:             else    # W przeciwnym przypadku:
31:                 fifo add(fictitious, queue) # dodajemy do kolejki piksel
fikcyjny
32:                 curdist←curdist + 1 # Zwiększamy zasięg zlewania
33:                 p ←fifo remove(queue) # przypisujemy piksel pobrany z kolejki
do p
34:             end if
35:         end if

36:         for all q ∈ NG(p) do # Dla wszystkich sąsiadów q piksela p
                # sprawdzamy czy piksel znajduje się w zasięgu zlewania i należy
do zbiornika lub jest pikselem wododziałowym
37:             if dist[q] < curdist and (lab[q] > 0 or lab[q] = wshed) then

38:                 if lab[q] > 0 then #jeśli q należy do zbiornika
                # jeśli p należy do zbiornika lub jest pikselem wododzialowym
39:                     if lab[p] = mask or lab[p] = wshed then
40:                         lab[p]←lab[q] # etykieta p taka jak etykieta q
                # Jeśli p nie należy do zbiornika  to ustawiamy jego etykietę
jako wshed
```

```
41:                     else if lab[p] != lab[q] then
42:                         lab[p]←wshed
43:                     end if
44:                 else if lab[p] = mask then
45:                     lab[p]←wshed
46:                 end if
47:             else if lab[q] = mask and dist[q] = 0 then
48:                 dist[q]←curdist + 1 # Zwiększamy obszar zlewania
49:                 fifo add(q, queue)  # Dodajemy q do kolejki
50:             end if
51:         end for #koniec iteracji po sąsiadach q piksela p
52:     end while
```

```
        #Szukamy nowych minimow dla danego poziomu jasności h
        # Dla każdego piksela o danym poziomie jasności
53:     for all p ∈ D with im[p] = h do
54:         dist[p]←0 # Resetujemy dystans
55:         if lab[p] = mask then # Czy piksel wewnątrz nowego minimum
56:             curlab←curlab + 1 # Tworzymy nową etykietę poprzez zwiększenie
poprzedniej
57:             fifo add(p, queue) # Dodajemy p do kolejki
58:             lab[p]←curlab       # przypisujemy obecną etykietę do macierzy
etykiet
59:             while not fifo empty(queue) do  # Do momentu kedy kolejka nie
jest pusta
60:                 q ←fifo remove(queue)   # Pobieramy piksel
61:                 for all r ∈ NG(q) do # Sprawdzamy sąsiadów r piksela q
62:                     if lab[r] = mask then # Jeśli r ma wartość mask
63:                         fifo add(r, queue) # Dodajemy r do kolejki
64:                         lab[r]←curlab # ustawiamy w macierzy etykiet dla r
aktualną                                            etykietę
65:                     end if
66:                 end for
67:             end while # kolejka jest pusta
68:         end if
69:     end for # koniec iteracji po kolejnych pikselach o danych poziomach
jasności
70: end for #koniec iteracji po odcieniach szarości
71: #Koniec zlewania
```

Wykonanie projektu

> Algorytm został zaimplementowany w języku **Python 3.7.0**.
> Do implementacji wykrzystane zostały następujące biblioteki (nazwa biblioteki oraz polecenie instalacji poprzez pip3):

- matplotlib - `pip3 install matplotlib`
- numpy - `pip3 install numpy`
- skimage - `pip3 install scikit-image`

Skrypt składa się z dwóch plików:

- **main.py** - główny plik skryptu. Utworzenie obiektu klasy `Image` oraz wywołanie jej metod.
  - Wejście:
    - Domyślnie bezargumentowo - segmentacja wykonywana na `coins.png`
    - Opcjonalnie argument wywołania programu - ścieżka do innego obrazu
  - Wyjście:
    - konsola - czasy wykonywania algorytmu wbudowanego oraz implementowaneg
    - W nowym oknie zestawienie obrazów będących wizualizacją wykonanych algorytmów
- **ImageClass.py** - Klasa `Image`, na której opiera się program.
  - `Image` - klasa przechowująca dane oraz implementację metod potrzebnych do wykonania algorytmu. Konstruktor przyjmuje ścieżkę do obrazu, na którym wykonana ma zostać segmentacja
    - `showImages` - metoda klasy `Image` odpowiadająca za wyświetlenie zestawienia obrazów w formie jednego.
    - `buildInWatershed` metoda klasy `Image` odpowiadająca za wykonanie budowanej segmentacji wododziałowej (z biblioteki skimage)
    - `prepareGrad` - metoda klasy `Image` odpowiadająca za przygotowanie gradientu używanego później w metodzie `watershed`
    - `watershed` - metoda klasy `Image` będąca implementacją omawianego algorytmu Vincent-Soille. Korzysta z metody `prepareGrad`

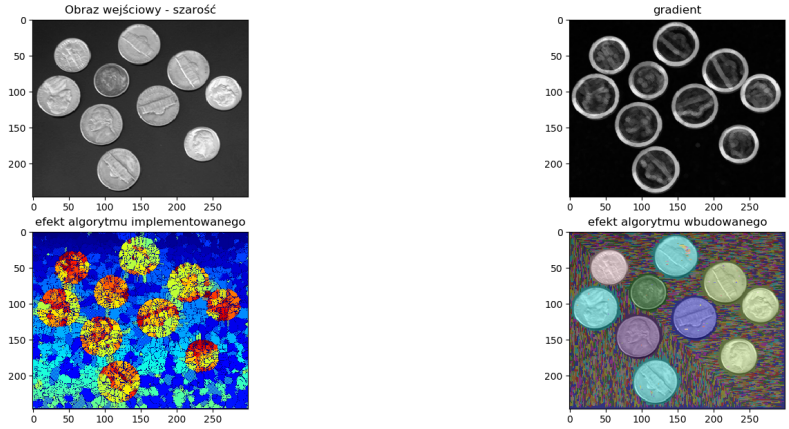
Efekt działania programu

W wyniku bezargumentowago uruchomienia programu otrzymujemy następujące rezultaty:

- Czas wykonywania algorytmu implementowanego: 10.35s
- Czas wykonywania algorytmu wbudowanego: 0.71s

Widzimy, że nasza implementacja algorytmu nie należy do szybkich. Znaczna różnica czasowa pomiędzy wbudowaną funkcją, a implementowaną wynika między innymi z różnych algorytmów.

Po wykonaniu programu zobaczymy okno z graficznymi rezultatami programu.



Widzimy, że zaimplementowany przez nas algorytm w danym przypadku spełnia swoje zadanie równie dobrze jak funkcja wbudowana w bibliotece skimage. Wszystkie kluczowe elementy zostały wydzielone. Efekty działania funkcji wbudowanej (prawy dół) jak i implementowanej (lewy dół) przez nas są bardzo podobne.