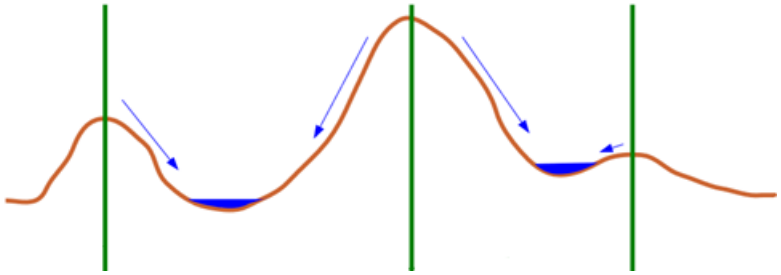
Segmentacja wododziałowa

Justyna Gacek, Magdalena Gajek

Opis problemu

Tematem naszego projektu była implementacja algorytmu realizującego segmentację wododziałową. Ogólnie rzecz biorąc segmentacja polega na podziale obrazu na części zwane regonami, które są jednorodne pod względem pewnych własności. Jedną z częściej stosowanych metod jest implementowana przez nas segmentacja wododziałowa. Jej idea została zaczerpnięta z geografii. Polega ona na wyznaczaniu linii rozdzielających hipotetyczne obszary zlewiskowe, które znajdują się wokół regionalnych minimów. Proces segmentacji jest tożsamy z zalewaniem wodą coraz wyżej położonych obszarów. Operacja zaczyna się od terenów położonych najniżej – czyli od najjaśniejszych pikseli, i trwa do momentu, aż wszystkie piksele zostaną zalane, czyli przydzielone do odpowiednich regionów. Poniżej zostały wyjaśnione podstawowe pojęcia:



Zlewiska to obszary, z których wody opadowe spływają do wspólnego zbiornika wodnego – na obrazie wynikowym odpowiadają wnętrzu podzielonych obszarów.

Wododziałem nazywamy zielone pionowe linie, które rozgraniczają tereny zlewisk – na obszarze wynikowym są to kontury podzielonych obrazów.

Wysokość terenu jest określeniem amplitudy gradientu obrazu.

Segmentacja wododziałowa wymaga odpowiedniego przygotowania obrazu. Obraz wejściowy należy przekonwertować do odcieni szarości. Następnie w zależności od obrazu należy wykonać różne operacje przygotowujące, aby wynik segmentacji był satysfakcjonujący. Ostatnim krokiem w przygotowywaniu obrazu jest utworzenie obrazu gradientowego, który jest utożsamiany z powierzchnią topograficzną. Niestety nie istnieje uniwersalny sposób na generowanie optymalnego obrazu gradientowego. W związku z tym ten sam obraz wejściowy może być niejednoznacznie określony przez obraz gradientowy.

Realizacja projektu

Algorytm segmentacji wododziałowej zaimplementowałyśmy przy użyciu programu Matlab. Jako dane wejściowe przyjmujemy zdjęcie, którego segmentację chcemy przeprowadzić. Przy realizacji projektu wykorzystywałyśmy następujące funkcje wbudowane:

- Imread() – wczytuje zdjęcie i zapisuje do zmiennej
- Rgb2gray()- przejście z RGB do odcieni szarości
- Strel()- tworzy obiekt strukturalny który wykorzystujemy przy tworzeniu gradientu
- Imerode()- operacja erozji
- Imreconstruct()- dokonuje rekonstrukcji erodowanego obrazu przy użyciu maski
- Imdilate()- operacja dylatacji
- Imcomplement()- dokonuje dopełnienia wartości poszczególnych pikseli na obrazie
- Size()- pobiera i zapisuje rozmiary zdjęcia
- Zeros()- tworzy macierze wypełnione zerami o zadanych wymiarach
- Unique()- usuwa powtarzające się elementy z tablicy
- Sort()- sortuje tablice
- label2rgb()- koloruje uzyskaną macierz z etykietami.

Do przechowywania współrzędnych poszczególnych pikseli wykorzystałyśmy kolejkę FIFO z biblioteki java.util. Obsługa kolejki wygląda następująco:

- LinkedList()- tworzy nową kolejkę
- Add()- dodaje nowy element do kolejki
- Remove()- usuwa i zwraca element z kolejki
- Size()- zwraca rozmiar kolejki

Opis algorytmu

Do implementacji wykorzystałyśmy pseudokod ze strony:
http://www.cs.rug.nl/~roe/publications/parwshed.pdf

```
1: procedure Watershed-by-Immersion
2: INPUT: digital grey scale image G = (D, E, im).
3: OUTPUT: labelled watershed image lab on D.
4: #define INIT − 1                    (∗ initial value of lab image ∗)
5: #define MASK − 2                    (∗ initial value at each level ∗)
6: #define WSHED 0                     (∗ label of the watershed pixels ∗)
7: #define FICTITIOUS (−1, −1)         (∗ fictitious pixel ∉ D ∗)
8: curlab ← 0                          (∗ curlab is the current label ∗)
9: fifo_init(queue)
10: for all p ∈ D do
11:   lab[p] ← INIT ; dist[p] ← 0      (∗ dist is a work image of distances ∗)
12: end for
13: SORT pixels in increasing order of grey values (minimum h_min, maximum h_max)
```

Na samym początku przygotowujemy zmienne pomocnicze:

- INIT = -1 - wartość początkowa dla wszystkich wartości w macierzy lab[]
- MASK = -2 - wartość początkowa dla każdego „zbiornika”
- WSHED = 0 - etykieta dla pikseli wododziałowych
- FICTITIOUS = -3 - wartość współrzędnych piksela nienależącego do zdjęcia wejściowego
- curlab = 0 - wartość inicjalizacyjna dla etykiet
- fifo_init(queue) – inicjalizacja kolejki FIFO

Pętla for (10-12) iteruje po wszystkich pikselach zdjęcia wejściowego. Wewnątrz pętli dla każdego piksela wartość etykiety zmieniamy na INIT, a wartość w macierzy odległości ustawiamy na 0. W kolejnym kroku sortujemy tabele z wartościami odcieni szarości.

```
16: for h = h_min to h_max do          (∗ Geodesic SKIZ of level h − 1 inside level h ∗)
17:   for all p ∈ D with im[p] = h do        (∗ mask all pixels at level h ∗)
18:                 (∗ these are directly accessible because of the sorting step ∗)
19:     lab[p] ← MASK
20:     if p has a neighbour q with (lab[q] > 0 or lab[q] = WSHED) then
21:       (∗ Initialize queue with neighbours at level h of current basins or watersheds ∗)
22:       dist[p] ← 1 ; fifo_add(p, queue)
23:     end if
24:   end for
```

Algorytm rozpoczyna pętla for iterująca po wszystkich odcieniach szarości zdjęcia wejściowego (16-69). Następnie iterujemy po wszystkich pikselach zawierających dany odcień szarości (17-24). Dla danego piksela (p) ustawiamy wartość etykiety na MASK i sprawdzamy czy piksel ten ma sąsiada (q) który spełnia któryś z warunków:

- lab[q] > 0- q jest zbiornikiem
- lab[q] = WSHED- q jest pikselem wododziałowym

Jeśli przynajmniej jeden z tych warunków jest spełniony to ustawiamy dist[p] = 1 i dodajemy

p do kolejki. W ten sposób dla każdego poziomu szarości otrzymamy piksele, które mają sąsiada posiadającego swoją etykietę.

```
25:   curdist ← 1 ; fifo_add(FICTITIOUS, queue)
26:   loop                (∗extend basins∗)
27:     p ← fifo_remove(queue)
28:     if p = FICTITIOUS then
29:       if fifo_empty(queue) then
30:         BREAK
31:       else
32:         fifo_add(FICTITIOUS, queue) ; curdist ← curdist + 1 ;
33:         p ← fifo_remove(queue)
34:       end if
35:     end if
36:     for all q ∈ N_G(p) do          (∗labelling p by inspecting neighbours∗)
37:       if dist[q] < curdist and (lab[q] > 0 or lab[q] = WSHED) then
38:         (∗q belongs to an existing basin or to watersheds∗)
39:         if lab[q] > 0 then
40:           if lab[p] = MASK or lab[p] = WSHED then
41:             lab[p] ← lab[q]
42:           else if lab[p] ≠ lab[q] then
43:             lab[p] ← WSHED
44:           end if
45:         else if lab[p] = MASK then
46:           lab[p] ← WSHED
47:         end if
48:       else if lab[q] = MASK and dist[q] = 0 then          (∗q is plateau pixel∗)
49:         dist[q] ← curdist + 1 ; fifo_add(q, queue)
50:       end if
51:     end for
52:   end loop
```

W linii 25 deklarujemy zasięg zalewania curdist = 1 i dodajemy do kolejki piksela, który znajduje się poza zdjęciem (piksel fikcyjny). Pętla loop służy do rozszerzania zalanego obszaru (linie 26-52). Ściągamy z kolejki piksel p, jeśli jest on fikcyjny to sprawdzamy czy kolejka jest pusta,

- jeśli tak, to przerywamy działanie pętli loop
- jeśli nie, dodajemy p do kolejki, zwiększamy wartość curdist i przypisujemy do p piksela pobranego z kolejki

Gdy upewnimy się, że piksel pobrany z kolejki nie wychodzi poza zakres, iterujemy po sąsiadach (q) danego piksela (p). Jeśli sąsiad q spełnia oba poniższe warunki (linia 37):

- dist[q] < curdist- piksel q znajduje się w zasięgu „zalewania”
- lab[q] > 0 lub lab[q] = WSHED - piksel q należy go jednego ze zbiorników lub jest pikselem wododziałowym

to możemy sprawdzić jego kolejne własności. Jeśli q należy do któregoś ze zbiorników (linia 39) i jeśli:

- lab[p] = MASK – p nie należy do żadnego zbiornika lub lab[p] = WSHED - p jest pikselem wododziałowym (linia 40) wtedy etykiecie p, przypisujemy wartość etykiety q (sąsiada). W przeciwnym wypadku sprawdzamy czy:
- lab[p] ≠ lab[q] - piksel p i jego sąsiad nie należą do tego samego zbiornika (linia 42) wtedy p oznaczamy jako piksela wododziałowego.

Jeśli powyższe warunki nie zostały spełnione i piksel p nie należy do żadnego zbiornika (linia 45) to ustawiamy jego etykietę na WSHED. Jeśli sąsiad q nie spełnia warunku z linii 37 to sprawdzamy czy spełnione są oba poniższe warunki (linia 48):

- lab[q] = MASK - sąsiad nie należy do żadnego zbiornika
- dist[q] = 0

Wtedy dodajemy q do kolejki, a wartość dist dla q ustawiamy na curdist + 1.

```
54:   for all p ∈ D with im[p] = h do
55:     dist[p] ← 0            (* reset distance to zero *)
56:     if lab[p] = MASK then          (* p is inside a new minimum *)
57:       curlab ← curlab + 1 ;        (* create new label *)
58:       fifo_add(p, queue) ; lab[p] ← curlab
59:       while not fifo_empty(queue) do
60:         q ← fifo_remove(queue)
61:         for all r ∈ N_G(q) do          (* inspect neighbours of q *)
62:           if lab[r] = MASK then
63:             fifo_add(r, queue) ; lab[r] ← curlab
64:           end if
65:         end for
66:       end while
67:     end if
68:   end for
69: end for
70: (* End Flooding *)
```

Następnie dla każdego piksela w obrazie o jasności równej h (54) resetowany jest dystans w tablicy dist (55). Kolejno sprawdzane jest czy piksel p znajduje się wewnątrz nowego minimum (56). Jeśli tak, tworzona jest nowa etykieta poprzez zwiększenie wartości zmiennej curlab, piksel p jest dodawany do kolejki, a w tablicy etykiet w miejscu piksela wpisywana jest wartość curlab, czyli obecnej etykiety (57-58).

Następnie za pomocą pętli while (59), która wykonuje się do momentu aż kolejka nie jest pusta, usuwany jest pierwszy piksel z kolejki (60). Dla ściągniętego piksela sprawdzani są wszyscy sąsiedzi po kolei (61). Jeśli sąsiad w tablicy etykiet lab ma wartość równą MASK, wtedy jest on dodawany do kolejki. A na jego miejsce w tablicy etykiet wpisywana jest

wartość bieżąca etykiety czyli curlab (62-64). Czynności te wykonywane są dla każdego z sąsiadów. Gdy kolejka jest pusta, pierwszy obieg pętli głównej idącej po odcieniach szarości jest zakończony.

Podział ról

Do przygotowania tego projektu zaangażowane były obie osoby. Wszelkie działania, zarówno implementacja algorytmu, jak i stworzenie dokumentacji wykonywane były wspólnie, dzięki czemu wszelkie niejasności i problemy mogły być rozwiązywane szybciej oraz efektywniej.

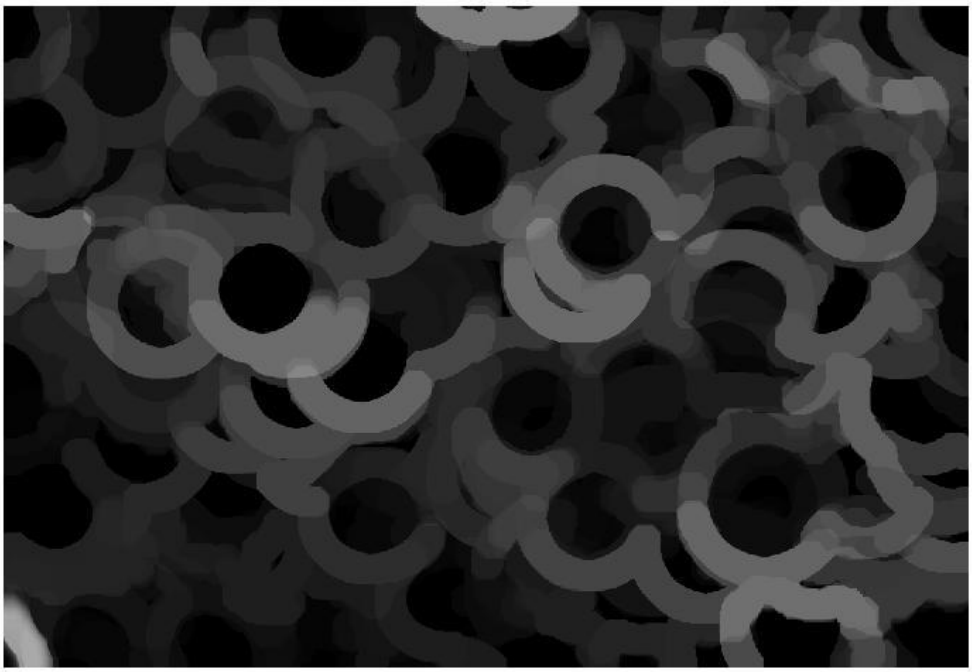
Wyniki

Algorytm przetestowałyśmy na gradiencie poniższego zdjęcia:

Zdjęcie 1. Monety

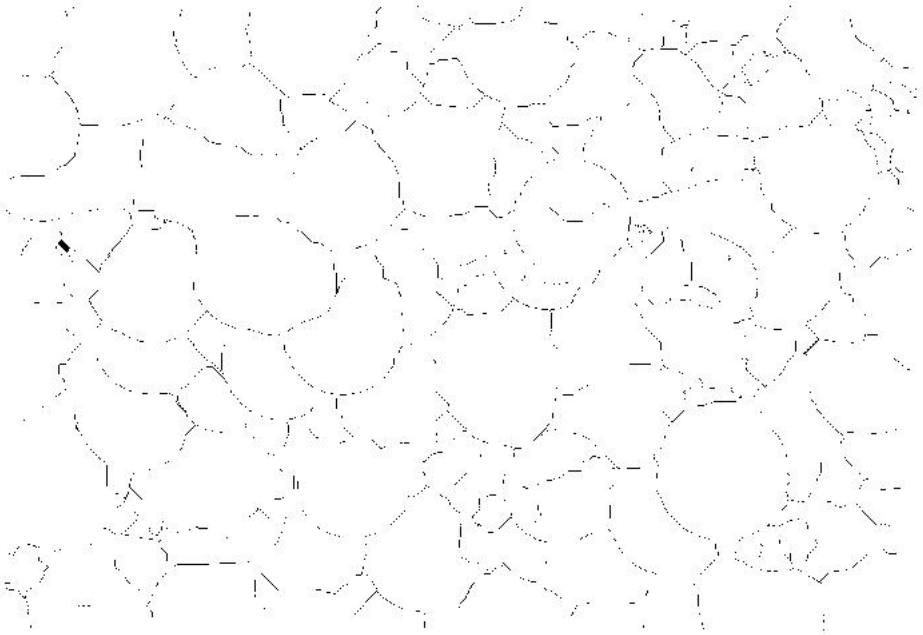
Do przygotowania gradientu posłużył nam poniższy fragment kodu:

```
I=imread('monety.jpg');
I = rgb2gray(I);

se = strel('disk', 17);
Ie = imerode(I, se);
Iobr = imreconstruct(Ie, I);

Iobrd = imdilate(Iobr, se);
Iobrcbr = imreconstruct(imcomplement(Iobrd), imcomplement(Iobr));
Iobrcbr = imcomplement(Iobrcbr);


Gradient = imdilate(Iobrcbr, se) - imerode(Iobrcbr, se);
```



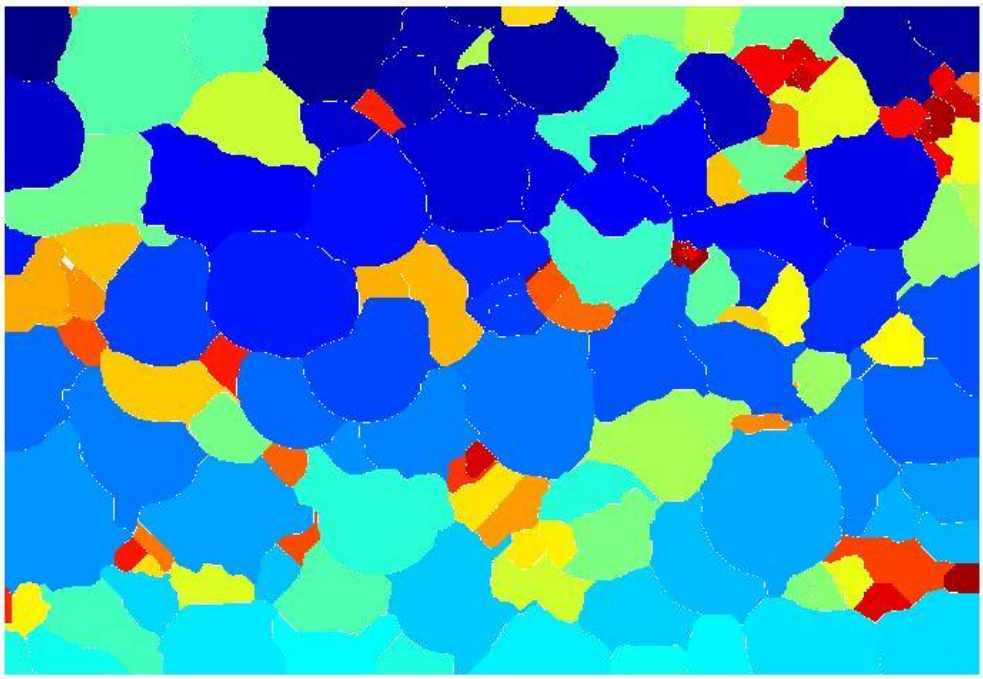
Zdjęcie 2. Gradient monet

Zdjęcie 2 zostało użyte to przetestowania działania zaimplementowanego algorytmu. Poniżej pokazane zostały uzyskane wyniki:



Zdjęcie 3. Obraz podzielony na elementy

Aby zobaczyć dokładnie jak algorytm wyodrębnił poszczególne monety pokolorujemy zdjęcie korzystając z funkcji wbudowanej label2rgb().



Zdjęcie 4. Pokolorowane zdjęcie wynikowe

Podsumowanie

W celu lepszej interpretacji efektu nałożone zostały na siebie kolorowy obraz wejściowy oraz powstały w wyniku segmentacji obraz z etykietami. Poniżej widoczny jest efekt działania korzystającego z wbudowanej funkcji **watershed** w matlabie:

Zdjęcie 5. Nałożenie oryginalnego zdjęcia i efektu segmentacji funkcją z Matlaba

Przestawiamy również efekt działania naszej implementacji algorytmu:

Zdjęcie 6. Nałożenie oryginalnego zdjęcia i efektu segmentacji naszą implementacją

Porównując efekt działania zaimplementowanego przez nas algorytmu oraz wynik działania funkcji wbudowanej w Matlabie możemy stwierdzić, że efekt naszej implementacji jest bardzo dobry. Duża część podzielonych obszarów jest identyczna w obu przypadkach. Nieznaczne różnice mogą wynikać z faktu, że użyte zostały różne algorytmy. Wbudowana funkcja używa algorytmu Fernanda Meyer'a, a my posłużyłyśmy się wersją Vincenta-Soille'a. Podsumowując przeprowadzona przez nas segmentacja w pełni spełniła swoją rolę. Widzimy, że pierwszoplanowe monety zostały w odpowiedni sposób wydzielone.