

Gliwice, 27.04.2022

Przetwarzanie Obrazów Cyfrowych

Wyznaczanie liczby barw w obrazie



Politechnika Śląska

Jakub Zeifert

1 Zliczanie barw unikalnych w RGB

- Przygotować minimum 2 obrazy barwne (O1, O2) .
- Za pomocą aplikacji POCLab/IrfanView lub innej wyznaczyć liczbę barw unikalnych (unikalnych trójek) w obrazie.
- Napisać skrypt w Matlab pozwalający na wyliczenie liczby barw unikalnych występujących w obrazie i sprawdzić poprawność działania z POCLab/IrfanView – wynik musi być identyczny.
- Uwaga: napisany skrypt nie może wykorzystywać funkcji typu: unique, permute, find. Zmierzyć czas potrzebny na wykonanie skryptu (polecenie tic, toc) – powinien być jak najkrótszy.

Program został zaimplementowany w języku Matlab.

```
1 clear ;
2
3 % Uploading Image
4 [ file ,path]=uigetfile ({'* .png' ;'* .BMP'} , 'Select an image') ;
5 img = imread ([ path , file ]) ;
6 img = imread ("img_008 .bmp") ;
7 img_2 = imread ("shrek_drzewa .png") ;
8 % RGB na HSV
9 hsv = rgb2hsv (img) ;
10 hsv_2 = rgb2hsv (img_2) ;
11 img_3 = hsv2rgb (hsv) ;
12 img_4 = hsv2rgb (hsv_2) ;
13 % Counting Colors
14 tic
15 rgb_colors = sum (find_rgb_colors (img , " ") , 'all ') ;
16 toc
17 tic
18 rgb_colors_2 = sum (find_rgb_colors (img_2 , " ") , 'all ') ;
19 toc
20 tic
21 rgb_colors_3 = sum (find_rgb_colors (img_3 , " normalized ") , 'all ') ;
22 toc
23 tic
24 rgb_colors_4 = sum (find_rgb_colors (img_4 , " normalized ") , 'all ') ;
25 toc
26 %% Plotting
27 figure ;
28 subplot (1 ,2 ,1 ) ;
imshow (img) ;
29 xlabel ("1 obraz ") ;
30 subplot (1 ,2 ,2 ) ;
imshow (img_2) ;
31 xlabel ('2 obraz ') ;
32
33 %% Functions
34
35
36
37
38 function result_img = find_rgb_colors (A ,target )
39
40 result_img = zeros (256 ,256 ,256) ;
41
42 [y , x , z] = size (A) ;
43
44 for i = 1 :1 :y
45     for j = 1 :1 :x
46         if target == ""
47             r = A (i ,j ,1) ;
48             g = A (i ,j ,2) ;
49             b = A (i ,j ,3) ;
50         elseif target == "normalized"
```

```

51 r = A(i,j,1)*256;
52 g = A(i,j,2)*256;
53 b = A(i,j,3)*256;
54 else
55     fprintf("%s is incorrect !!!", target);
56     break
57 end
58 if result_img(uint16(r)+1,uint16(g)+1,uint16(b)+1) == 0
59     result_img(uint16(r)+1,uint16(g)+1,uint16(b)+1) = 1;
60 end
61 end
62 end
63
64 end

```

zad1.m

Program tworzy szcześcią zer, gdzie można zmieścić wszystkie możliwe kombinacje kolorów **RGB**. Następnie wpisuje wartość 1 w każde miejsce gdzie dana kombinacja się znalazła. Następnie zliczana jest suma jedynek w sześciianie, a więc ilość unikalnych kombinacji składowych r,g,b.



Liczba Barw według programu : 98541,
Wartość według programu POC Lab: 98541



Liczba Barw według programu : 168175,
Wartość według programu POC Lab: 168175

Rysunek 1: Liczby barw unikalnych dla poszczególnych obrazów RGB

Poszczególne czasy działania programu dla każdego z obrazów:

- Gruszka: 0.106583s, $RGB_{normalized} = 0.285095s$
- Shrek: 0.074891s, $RGB_{normalized} = 0.216023s$

Widać, że zliczanie barw odbywa się sprawnie. Ciekawym spostrzeżeniem jest fakt, że obraz posiadający mniej barw potrzebował więcej czasu, aby przeliczyć. Czasy dla znormalizowanego obrazu są dłuższe niż dla nieznormalizowanego. Wynika to najprawdopodobniej z przemnożenia każdej wartości przez 256. Natomiast liczba kolorów jest stała dla każdego z obrazów.

2 Zliczanie barw w HSV

- Dla obrazów O1 i O2 wykonać transformaty do przestrzeni barw HSV (rgb2hsv)
- Dostosować skrypt napisany w Zad. 1 do zliczania barw unikalnych w HSV
- Sprawdzić działanie skryptu, czy liczba barw unikalnych po transformacie do HSV się zmienia?
Wskazówka: wykonując transformację O1 z RGB do HSV ($O1HSV=rgb2hsv(O1)$) a następnie odwrotną transformację z HSV do RGB ($O1RGB=hsv2rgb(O1HSV)$) jaka jest zależność pomiędzy obrazami O1 i O1RGB ??

Program został zaimplementowany w języku Matlab.

```
1 clear ;
2
3 % Uploading Image
4 [ file ,path]=uigetfile ({'* .png' ;'* .BMP'} , 'Select an image') ;
5 % img = imread ([ path , file ]) ;
6 img = imread ("img_008.bmp") ;
7 img_2 = imread ("shrek_drzewa.png") ;
8 hsv = rgb2hsv (img) ;
9 hsv_2 = rgb2hsv (img_2) ;
10 tic
11 hsv_amount = sum (find_hsv_colors (hsv) , 'all ') ;
12 toc
13 tic
14 hsv_amount_2 = sum (find_hsv_colors (hsv_2) , 'all ') ;
15 toc
16
17
18
19 %% Plotting
20 figure ;
21 subplot (1,2,1) ;
22 imshow (img) ;
23 xlabel ("1 obraz ") ;
24 subplot (1,2,2) ;
25 imshow (img_2) ;
26 xlabel ('2 obraz ') ;
27
28 %% Functions
29
30 function result_img = find_hsv_colors (img)
31
32     result_img = zeros (1700,256,256) ;
33     [x,y,z] = size (img) ;
34
35     for i=1:1:x
36         for j=1:1:y
37             hue = round (img (i,j,1)*1700) ;
38             sat = round (img (i,j,2)*256) ;
39             val = round (img (i,j,3)*256) ;
40             result_img (uint16 (hue)+1,uint16 (sat)+1,uint16 (val)+1) = 1;
41         end
42     end
43 end
```

zad2.m

Program został lekko zmodyfikowany względem pierwszego. Różnica jest w denormalizacji wartości w systemie **HSV** oraz uwzględnianie szerokiego zakresu **HUE** do 1700, a więc jego wartość została poszerzona, gdyż barwy mogą się znacznie bardziej różnić niż o jeden stopień (co działałoby się przy przemnożeniu przez 360).



Liczba Barw według programu : 98541,
Wartość według programu POC Lab: 98541



Liczba Barw według programu : 168175,
Wartość według programu POC Lab: 168175

Rysunek 2: Liczby barw unikalnych dla poszczególnych obrazów HSV

Poszczególne czasy działania programu dla każdego z obrazów:

- Gruszka:RGB 1.075455s
- Shrek: 0.872410s

Liczenie barw w systemie HSV zajęła więcej czasu niż dla RGB lecz nieznacząco. Ponownie przetworzenie obrazu, który posiada więcej barw odbyło się szybciej.

3 kwantyzacja w RGB

- Zaimplementować metodę kwantyzacji barwy z paletą stałą polegającą na odpowiednim podziale przestrzeni RGB. Zastosować kolejno podziały dla składowych RGB: 2x2x2, 4x4x4, 4x6x4, 8x8x4.
- Wyznaczyć liczbę barw unikalnych oraz wskaźnik PSNR. Czy wyznaczona liczba barw unikalnych pokrywa się z założonym podziałem przestrzeni barw? Jeżeli nie to dlaczego?

Program został zaimplementowany w języku Matlab.

```

1 clc ; clear ;
2
3 % Uploading Image
4 [ file ,path]=uigetfile ({'* .png' ;'* .BMP'} , 'Select an image ') ;
5 img = imread ([ path , file ]) ;
6 % img = imread ("img_008.bmp") ;
7 img = imread ("shrek_drzewa.png") ;
8
9
10 % quantization
11 rgb_colors_full = sum (find_rgb_colors (img , " ") , ' all ') ;
12 img_8 = quantize_rgb (img , 2 , 2 , 2 ) ;
13 rgb_colors_8 = sum (find_rgb_colors (img_8 , " ") , ' all ') ;
14 img_64 = quantize_rgb (img , 4 , 4 , 4 ) ;
15 rgb_colors_64 = sum (find_rgb_colors (img_64 , " ") , ' all ') ;
16 img_128 = quantize_rgb (img , 4 , 6 , 4 ) ;
17 rgb_colors_128 = sum (find_rgb_colors (img_128 , " ") , ' all ') ;
18 img_256 = quantize_rgb (img , 6 , 6 , 6 ) ;
19 rgb_colors_256 = sum (find_rgb_colors (img_256 , " ") , ' all ') ;
20
21 imwrite (img , " photos /img .png ")
22 imwrite (img_8 , " photos /img8 .png ") ;
23 imwrite (img_64 , " photos /img64 .png ") ;
24 imwrite (img_128 , " photos /img128 .png ") ;

```

```

25 imwrite(img_256,"photos/img256.png");
26
27 psnr_8 = psnr(img_8,img);
28 psnr_64 = psnr(img_64,img);
29 psnr_128 = psnr(img_128,img);
30 psnr_256 = psnr(img_256,img);
31
32 %% Plotting
33 figure;
34 subplot(1,5,2);
35 imshow(img_8);
36 xlabel("8 barw");
37 subplot(1,5,1);
38 imshow(img);
39 xlabel('orygina Ć');
40 subplot(1,5,3);
41 imshow(img_64);
42 xlabel('64 barwy');
43 subplot(1,5,4);
44 imshow(img_128);
45 xlabel('128 barw');
46 subplot(1,5,5);
47 imshow(img_256);
48 xlabel('256 barw');

49
50
51 % imwrite(result,"quantized.png");
52
53 %% Functions
54 function result_img = find_rgb_colors(A,target)
55
56     result_img = zeros(256,256,256);
57
58     [y, x, z] = size(A);
59
60     for i = 1:1:y
61         for j = 1:1:x
62             if target == ""
63                 r = A(i,j,1);
64                 g = A(i,j,2);
65                 b = A(i,j,3);
66             elseif target == "normalized"
67                 r = A(i,j,1)*256;
68                 g = A(i,j,2)*256;
69                 b = A(i,j,3)*256;
70             else
71                 fprintf("%s is incorrect",target);
72             end
73             result_img(uint16(r)+1,uint16(g)+1,uint16(b)+1) = 1;
74         end
75     end
76 end

77 function result_img = quantize_rgb(img, r, g, b)
78     [x,y,z] = size(img);
79     result_img = zeros(x,y,z, 'uint8');
80     red_div = 256/r;
81     green_div = 256/g;
82     blue_div = 256/b;
83
84     for i = 1:1:x
85         for j = 1:1:y
86             result_img(i,j,1) = uint8(floor(double(img(i,j,1))/red_div)*red_div +
87                                         red_div/2);
88             result_img(i,j,2) = uint8(floor(double(img(i,j,2))/green_div)*green_div +

```

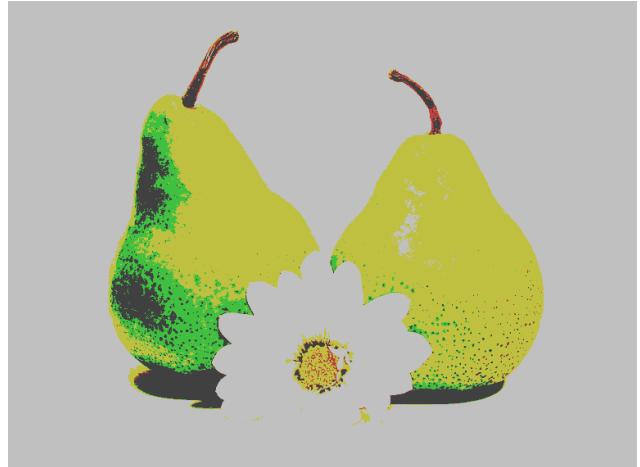
```
89     green_div/2);
90     result_img(i,j,3) = uint8(floor(double(img(i,j,3))/blue_div)*blue_div +
91     blue_div/2);
92 end
```

zad3.m

Każda wartość z poszczególnych pikseli jest wyznaczana poprzez dzielenie pełnego zakresu na konkretnie wybraną ilość przedziałów, a następnie przypisywana wartość równa wartości środkowej przedziału, w którym znalazła się dana wartość sprawdzana.



Oryginał
Liczba Barw: 98541



2x2x2
Liczba Barw: 6
PSNR = 13.410db



4x4x4
Liczba Barw: 25
PSNR = 19.587db



4x6x4
Liczba Barw: 32
PSNR = 20.533db



8x8x4
Liczba Barw: 63
PSNR = 23.457db

Rysunek 3: Obrazy swkantyzowane RGB

Bardzo dobrze widać działanie kwantyzacji. Poza widoczną zmianą ilości barw wraz ze wzrostem ich dostępności możemy zauważać znaczną zmianę tła gruszek, która wraca do normalności wraz ze wzrostem ilości barw. Wynika to z ustawiania zawsze średniej wartości z danego przedziału. W takim rozwiążaniu po kwantyzacji nie będzie możliwe otrzymanie "czystej" bieli (255,255,255) oraz czarnego (0,0,0). wskaźnik **PSNR** wzrasta wraz z ilością dostępnych barw, co nie jest specjalnym zaskoczeniem.



Oryginał
Liczba Barw: 168175



2x2x2
Liczba Barw: 8
PSNR = 15.788db



4x4x4
Liczba Barw: 38
PSNR = 22.349db



4x6x4
Liczba Barw: 51
PSNR = 23.124db



8x8x4
Liczba Barw: 106
PSNR = 26.166db

Rysunek 4: Obrazy swkantyzowane RGB

Tak jak na porzednim obrazie widać porawę różnicy między poszczególnymi częściami obrazu wraz ze wzrostem wartości dostępnych unikalnych barw. Tak samo wzrasta wskaźnik **PSNR**. W przypadku kwantyzacji na 8 barw obraz wykorzystał wszystkie dostępne. PSNR jest wyższy niż dla obrazu pierwszego, co jest spowodowane najprawdopodobniej przez tło, które zostało tak popskute.

4 kwantyzacja w HSV

- Zaimplementować metodę kwantyzacji barwy z paletą stałą polegającą na odpowiednim podziałe przestrzeni HSV. Zastosować kolejno podziały dla składowych HSV: 2x2x2, 4x4x4, 4x6x4, 10x5x5+6 (zakładamy 6 barw achromatycznych dla których nasycenie jest zerowe a odcień jest

niezdefiniowany).

- UWAGA: kwantyzacja w przestrzeni HSV wymaga wcześniejszej transformaty (rgb2HSV). Sam podział przestrzeni i redukcja barw jest realizowany w HSV. W celu właściwego wyświetlenia wyniku kwantyzacji i wyznaczenia PSNR i liczby barw unikalnych należy powrócić do RGB (HSV2RGB).
- Wyznaczyć liczbę barw unikalnych oraz wskaźnik PSNR.

Program został zaimplementowany w języku Matlab.

```
1 clc ; clear ;
2
3 % Uploading Image
4 [ file ,path]=uigetfile ({'* .png' ;'* .BMP'} , 'Select an image') ;
5 img = imread ([ path , file ]) ;
6 img = imread ("img_008 .bmp") ;
7 % img = imread ("shrek_drzawa .png") ;
8 img = rgb2HSV (img) ;
9
10 % quantization
11 rgb _ colors _ full = sum (find _ HSV _ colors (img) , 'all ') ;
12 img _ 8 = quantize _ HSV (img , 2 , 2 , 2 , 0 ) ;
13 rgb _ colors _ 8 = sum (find _ HSV _ colors (img _ 8) , 'all ') ;
14 img _ 64 = quantize _ HSV (img , 4 , 4 , 4 , 0 ) ;
15 rgb _ colors _ 64 = sum (find _ HSV _ colors (img _ 64) , 'all ') ;
16 img _ 128 = quantize _ HSV (img , 4 , 6 , 4 , 0 ) ;
17 rgb _ colors _ 128 = sum (find _ HSV _ colors (img _ 128) , 'all ') ;
18 img _ 256 = quantize _ HSV (img , 10 , 5 , 5 , 6 ) ;
19 rgb _ colors _ 256 = sum (find _ HSV _ colors (img _ 256) , 'all ') ;
20
21
22 img = HSV2RGB (img) ;
23 psnr _ 8 = psnr (img _ 8 , img) ;
24 psnr _ 64 = psnr (img _ 64 , img) ;
25 psnr _ 128 = psnr (img _ 128 , img) ;
26 psnr _ 256 = psnr (img _ 256 , img) ;
27
28
29 imwrite (img , "photos /img .png")
30 imwrite (img _ 8 , "photos /img8 .png") ;
31 imwrite (img _ 64 , "photos /img64 .png") ;
32 imwrite (img _ 128 , "photos /img128 .png") ;
33 imwrite (img _ 256 , "photos /img256 .png") ;
34 %% Plotting
35 figure ;
36 subplot (1 , 5 , 2 ) ;
37 imshow (img _ 8 ) ;
38 xlabel ('8 barw') ;
39 subplot (1 , 5 , 1 ) ;
40 imshow (img) ;
41 xlabel ('orygina Ć') ;
42 subplot (1 , 5 , 3 ) ;
43 imshow (img _ 64 ) ;
44 xlabel ('64 barwy') ;
45 subplot (1 , 5 , 4 ) ;
46 imshow (img _ 128 ) ;
47 xlabel ('128 barw') ;
48 subplot (1 , 5 , 5 ) ;
49 imshow (img _ 256 ) ;
50 xlabel ('256 barw') ;
51
52
53
54 %% Functions
55 function result _ img = find _ HSV _ colors (img)
```

```

56
57 result_img = zeros(1700,256,256);
58 [x,y,z] = size(img);
59
60 for i=1:x
61     for j=1:y
62         hue = round(img(i,j,1)*1700);
63         sat = round(img(i,j,2)*256);
64         val = round(img(i,j,3)*256);
65         result_img(uint16(hue)+1,uint16(sat)+1,uint16(val)+1) = 1;
66     end
67 end
68
69 function result_img = quantize_hsv(img, h, s, v, chroma)
70 [x,y,z] = size(img);
71 result_img = zeros(x,y,z);
72 h_div = 1/h;
73 s_div = 1/s;
74 v_div = 1/v;
75 if chroma ~= 0
76     chroma_div = 1/chroma;
77 end
78 for i = 1:x
79     for j = 1:y
80         if img(i,j,2) < 0.1 && chroma ~= 0
81             result_img(i,j,1) = 0;
82             result_img(i,j,2) = 0;
83             result_img(i,j,3) = floor(img(i,j,3)/chroma_div) * (chroma_div) + (
84                 chroma_div/2);
85         else
86             result_img(i,j,1) = floor(img(i,j,1)/h_div) * (h_div) + (h_div/2);
87             result_img(i,j,2) = floor(img(i,j,2)/s_div) * (s_div) + (s_div/2);
88             result_img(i,j,3) = floor(img(i,j,3)/v_div) * (v_div) + (v_div/2);
89         end
90         if result_img(i,j,1) >= 1
91             result_img(i,j,1) = floor(0.9/h_div) * (h_div) + (h_div/2);
92         end
93         if result_img(i,j,2) >= 1
94             result_img(i,j,2) = floor(0.9/s_div) * (s_div) + (s_div/2);
95         end
96         if result_img(i,j,3) >= 1
97             result_img(i,j,3) = floor(0.9/v_div) * (v_div) + (v_div/2);
98         end
99         if result_img(i,j,1) <= (h_div/2)
100            result_img(i,j,1) = (h_div/2);
101        end
102        if result_img(i,j,2) <= (s_div/2)
103            result_img(i,j,2) = (s_div/2);
104        end
105        if result_img(i,j,3) <= (v_div/2)
106            result_img(i,j,3) = (v_div/2);
107        end
108    end
109 end
110 result_img = round(abs(hsv2rgb(result_img)),3,'significant');
111 end

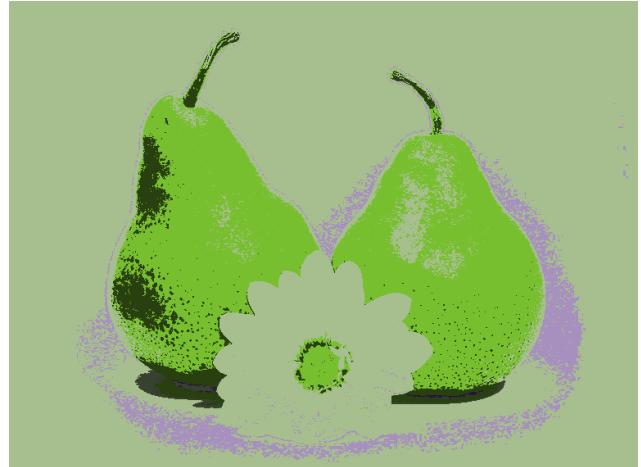
```

zad4.m

Program działa identycznie jak w zdaniu czwartym z niewielkimi zmianami. Wprowadzone zostało między innymi naprawa przedziałów, gdyż wartości potrafiły wynosić więcej niż skala pozwalała lub mniej niż powinna. również rozważane są barwy achromatyczne.



Oryginał
Liczba Barw: 98541



2x2x2
Liczba Barw: 7
PSNR = 10.612db



4x4x4
Liczba Barw: 37
PSNR = 16.459db



4x6x4
Liczba Barw: 51
PSNR = 17.339db



10x5x5 +6
Liczba Barw: 106
PSNR = 17.960db

Rysunek 5: Obrazy swkantyzowane HSV



Oryginał
Liczba Barw: 168175



2x2x2
Liczba Barw: 8
PSNR = 17.106db



4x4x4
Liczba Barw: 55
PSNR = 21.831db



4x6x4
Liczba Barw: 82
PSNR = 22.00db



10x5x5 +6
Liczba Barw: 178
PSNR = 24.583db

Rysunek 6: Obrazy swkantyzowane HSV

W przypadku obu zestawów pojawia się problem przesunięcia kolorów. Kolory są odwzorowywane odpowiednio jeśli chodzi o ilość ograniczonych kolorów, lecz wartości (najprawdopodobniej hue) są często przesunięte. Nie wiadomo czy błąd wynika z implementacji, czy z natury zachowywania się przekształceń z rgb na hsv i odwrotnie czy może pewna zależność nie została uwzględniona. Wartości PSNR rosną wraz ze wzrostem dostępnych barw, co również nie powinno zaskakiwać.