ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE
KATEDRA GEOMATIKY

název předmětu

GEOINFORMATIKA

číslo úlohy	název úlohy				
1	JPEG komprese a dekomprese rastru				
školní rok	studijní skup.	číslo zadání	Zpracovali:	datum	klasifikace
2023	C-101	-	Josef Jehlička Kateřina Chromá Štěpán Šedivý	8.12. 2023	

TECHNICKÁ ZPRÁVA

ZADÁNÍ:

Úkolem bylo implementovat algoritmus pro JPEG kompresi a rastru zahrnující tyto fáze:

- transformaci do YC_BC_R modelu,
- diskrétní kosinovou transformaci,
- kvantizaci koeficientů,

Kompresní algoritmus bylo nutné otestovat na různých typech rastru, kterými jsou: rastr v odstínech šedi, barevný rastr, vhodného rozlišení a velikosti s různými hodnotami faktoru komprese
$q = 10, 50, 70.$

Pro každou variantu je zadáno vypočíst střední kvadratickou odchylku *m* jednotlivých *RGB* složek.

$$m = \sqrt{\left(\frac{\sum_{i=0}^{m*n}(z-z')^2}{m*n}\right)}.$$

Úkolem je též zhodnotit vhodnost jednotlivých rastrů pro JPEG kompresi.

POPIS A ROZBOR PROBLÉMU:

JPG komprese

Nejprve je potřeba převést *RGB (red, green, blue)* do barevného prostoru YC_BC_R *(luma, blue and red difference chroma component)* pomocí definovaných konstant.

$$\begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} = \begin{pmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1687 & -0.3313 & 0.5000 \\ 0.5000 & -0.4187 & -0.0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}$$

Dále bylo pro použití diskrétní kosinové transformace nutné vstupní rastr převzorkovat na submatice 8x8, které do ní budou postupně vstupovat. Každý tento blok je podroben zmiňované DCT transformaci, což vytváří množinu koeficientů, které reprezentují frekvenční charakteristiky bloku. DCT je definována rovnicí níže.

$$F(u,v) = \frac{1}{4}C(u) * C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) * \cos\frac{(2x+1)u\pi}{16} * \frac{(2y+1)v\pi}{16}\right]$$

Kde

$$C(u) = \begin{cases} \frac{\sqrt{2}}{2}, & u = 0, \\ 1, & u \neq 0. \end{cases},$$

$$C(v) = \begin{cases} \frac{\sqrt{2}}{2}, & v = 0, \\ 1, & v \neq 0. \end{cases}.$$

Výsledné koeficienty jsou zaokrouhleny, což je s kvantizací jeden ze ztrátových faktorů JPEG komprese. Dále je provedena samotná kvantizace pomocí definovaných kvantizačních matic. Kvantizace způsobuje, že vyšší frekvenční složky jsou kvantizovány s nižší přesností než nižší frekvenční složky. Použité kvantizační matice vypadají následovně:

$$F_Q(u,v) = \frac{F(u,v)}{Q(u,v)},$$

Kde *Q(u,v)* je rozdílné pro složky *Y* a *C:*

$$Q(u,v)_{50}^{Y} = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 87 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 26 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}$$

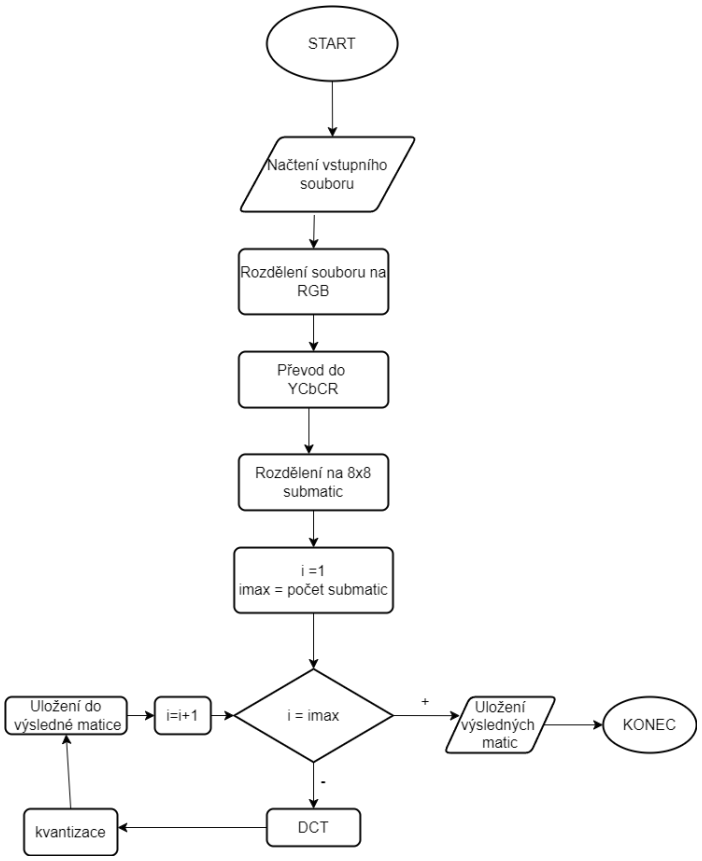
$$Q(u,v)_{50}^{C} = \begin{pmatrix} 17 & 18 & 24 & 47 & 66 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 69 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}$$

$$Q(u,v) = \frac{50 * Q(u,v)_{50}}{q}$$

Výsledkem jsou kompresované submatice pro jednotlivé kanály barevného prostoru YC_BC_R.

IMPLEMETACE:

JPEG komprese byla provedena v softwaru MATLAB dle následujícího vývojového diagramu:



Načtení vstupního souboru bylo provedeno pomocí funkce *imread* a zobrazen pomocí *imshow*:

```
input_image = imread('image2.bmp');
imshow(input_image,[0 80]);
```

Vstupní rastrové soubory 8bitové barevné hloubky mají zvolené rozměry 256x256 pixelů pro urychlený proces výpočtu.
Takto načtený obrázek byl rozdělen na RGB složky:

```
R = double(input_image(:,:,1));
G = double(input_image(:,:,2));
B = double(input_image(:,:,3));
```

RGB bylo převedeno na YCbCr:

```
Y=0.2990*R+0.5870*G+0.1140*B;
Cb=-0.1687*R-0.3313*G+0.5000*B+128;
Cr=0.5000*R-0.4187*G-0.0813*B+128;
```

Poté byla vytvořena smyčka rozdělující matice na 8x8 pixelů, v níž byla provedena DCT, kvantizace a tvorba kompresovaných matic:

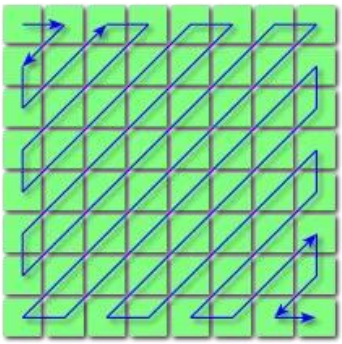
```
% Initialize transformed components

[m, n] = size(R);
Y_transformed = Y;
Cb_transformed = Cb;
Cr_transformed = Cr;


for i = 1:8:m-7
    for j = 1:8:n-7

        Y_submatrix = Y_transformed(i:i+7, j:j+7);
        Cb_submatrix = Cb_transformed(i:i+7, j:j+7);
        Cr_submatrix = Cr_transformed(i:i+7, j:j+7);

        %  DCT
        Y_submatrix_dct = mydct(Y_submatrix);
        Cb_submatrix_dct = mydct(Cb_submatrix);
        Cr_submatrix_dct = mydct(Cr_submatrix);


        Y_submatrix_dct = round(Y_submatrix_dct);
        Cb_submatrix_dct = round(Cb_submatrix_dct);
        Cr_submatrix_dct = round(Cr_submatrix_dct);



        Y_quantization_matrix_Y = 50 * quantization_matrix_Y / compression_factor;
        CbCr_quantization_matrix_C = 50 * quantization_matrix_CbCr / compression_factor;

        % Quantization
        Y_quantized = round(Y_submatrix_dct ./ Y_quantization_matrix_Y);
        Cb_quantized = round(Cb_submatrix_dct ./ CbCr_quantization_matrix_C);
        Cr_quantized = round(Cr_submatrix_dct ./ CbCr_quantization_matrix_C);

        % Update the compressed submatrices
        Y_transformed(i:i+7, j:j+7) = Y_quantized;
        Cb_transformed(i:i+7, j:j+7) = Cb_quantized;
        Cr_transformed(i:i+7, j:j+7) = Cr_quantized;
    end
end
```

Samotná funkce provádějící DCT pojmenována *mydct* přijímá na vstupu 8x8 matici a její výstup je transformovaná matice obsahující DCT koeficienty. Dvojitá vnořená smyčka (u, v) prochází všechny možné frekvenční koeficienty v 8x8 bloku. Pro každou hodnotu *u* a *v* jsou definovány váhy *Cu* a *Cv*. Pokud *u* a *v* jsou rovny 0, váhy jsou nastaveny jako polovina odmocniny ze dvou, jinak jsou nastaveny na 1. Další dvojitá vnořená smyčka pro *x* a *y* prochází všechny pixely v bloku. Pro každý pixel se vypočítá součet podle vzorce pro DCT. Výsledek se přičítá k proměnné *sumResult*. Vypočítaný součet je uložen jako hodnota odpovídajícího DCT koeficientu. Funkce vypadá takto:

```
function [resultingTransform] = mydct(inputMatrix)

% Custom DCT
resultingTransform = inputMatrix;

for u = 0:7
    for v = 0:7

        % Define  Cu, Cv
        if (u == 0)
            Cu = sqrt(2) / 2;
        else
            Cu = 1;
        end

        if (v == 0)
            Cv = sqrt(2) / 2;
        else
            Cv = 1;
        end


        % for each pixel
        sumResult = 0;
        for x = 0:7
            for y = 0:7
                sumResult = sumResult + (1 / 4) * Cu * Cv * inputMatrix(x + 1, y + 1) * ...
                    cos((2 * x + 1) * u * pi / 16) * cos((2 * y + 1) * v * pi / 16);
            end
        end
        %  DCT coefficients
        resultingTransform(u + 1, v + 1) = sumResult;
    end
end
end
```

BONUSOVÉ ÚLOHY:

1. Konverze pixelů do ZIG-ZAG sekvencí
   Kompresované barevné složky YC_BC_R je třeba převést z matic do jedné řady pomocí Zig-zag sekvence. Tím byla převedena 2D data na 1D. Funkce konverze je vyznačena na následujícím obrázku.



Pro tento účel byla vytvořena funkce *zigzagToOneLine*, která funguje takto:

1) Inicializace proměnných:

- Získá rozměry vstupní matice *inputArray* pomocí size a přiřadí je do proměnných *rows* a *cols*.
- Inicializuje pole *oneLineArray* o délce *rows * cols* plné nul.

2) Inicializace pozice a směru:

- Nastaví počáteční pozici na první řádek a první sloupec matice *(row = 1, col = 1).*
- Inicializuje proměnnou *direction* na -1, což znamená, že začíná směrem nahoru.

3) Cyklus procházení prvků:

- Používá cyklus pro iteraci přes všechny prvky matice.
- Přiřazuje hodnotu aktuálního prvku do odpovídající pozice v jednorozměrném poli *oneLineArray*.

4) Aktualizace pozice a směru:

- Podle směru (*direction*) aktualizuje pozici pro další prvek v matici.
- Při směru -1 (nahoru):
  - Pokud je možné pohybovat se nahoru a doleva, aktualizuje pozici.
  - Pokud narazí na horní hranici matice, přepíná směr dolů.
  - Pokud není možné pohybovat se nahoru a doleva, posouvá se doprava a přepíná směr na dolů.
- Při směru 1 (dolů):
  - Pokud je možné pohybovat se dolů a doprava, aktualizuje pozici.
  - Pokud narazí na spodní hranici matice, přepíná směr na nahoru.
  - Pokud není možné pohybovat se dolů a doprava, posouvá se doprava a přepíná směr na nahoru.

5) Návrat výsledného pole:

- Funkce vrátí toto pole *oneLineArray* jako výstup.

Funkce vypadá následovně:

```
function oneLineArray = zigzagToOneLine(inputArray)
    [rows, cols] = size(inputArray);
    oneLineArray = zeros(1, rows * cols);


    row = 1;
    col = 1;
    direction = -1; % Start with up

    for index = 1:(rows * cols)
        % Assign element to the output array
        oneLineArray(index) = inputArray(row, col);

        % Update  position with direction
        if direction == -1
            % Check for boundary
            if row > 1 && col < cols
                row = row - 1;
                col = col + 1;
            elseif col == cols
                row = row + 1;
                direction = 1; % Change to down
            else
                col = col + 1;
                direction = 1; % Change to down
            end
```

```
        else
                    % Check for boundary
                    if col > 1 && row < rows
                        row = row + 1;
                        col = col - 1;
                    elseif row == rows
                        col = col + 1;
                        direction = -1; % Change to upw
                    else
                        row = row + 1;
                        direction = -1; % Change to up
                    end
                end
            end
        end
```

Pro účely dekomprese byly hodnoty barevných složek z jedné řady (1D) převedeny zpět na původní matice(2D) pomocí obdobně fungující funkce *zigzagMatrix* (viz příloha).

2. JPG dekomprese

JPG dekomprese postupuje po submaticích stejně jako JPG komprese.

```
Y_submatrix = Y_transformed(i:i+7, j:j+7);
Cb_submatrix = Cb_transformed(i:i+7, j:j+7);
Cr_submatrix = Cr_transformed(i:i+7, j:j+7);
```

Barevné kanály jsou uvnitř smyčky pro celý rastr jsou nejprve dekvantizovány:

```
Y_dequantized = round(Y_submatrix .* Y_quantization_matrix_Y);
Cb_dequantized = round(Cb_submatrix .* CbCr_quantization_matrix_C);
Cr_dequantized = round(Cr_submatrix .* CbCr_quantization_matrix_C);
```

Na dekvantizovaných kanálech je provedena inverzní diskrétní kosinová transformace dle vztahů:

$$F(x,y) = \frac{1}{4}\left[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u) * C(v) f(u,v) * \cos\frac{(2x+1)u\pi}{16} * \frac{(2y+1)v\pi}{16}\right]$$

kde

$$C(u) = \begin{Bmatrix} \frac{\sqrt{2}}{2}, & u = 0, \\ 1, & u \neq 0. \end{Bmatrix},$$

$$C(v) = \begin{Bmatrix} \frac{\sqrt{2}}{2}, & v = 0, \\ 1, & v \neq 0. \end{Bmatrix}.$$

Ta byla implementována funkcí *myidct*, která funguje obdobně jako dříve popsaná funkce *mydct*. Nejprve vnější smyčky pro *x* a *y* iterují přes každý pixel v 8x8 bloku. Vnitřní smyčky pro *u* a *v* terují přes všechny možné frekvenční složky v 8x8 bloku. Koeficienty *Cu* a *Cv* se vypočítají na základě *u* a *v* obdobně jako u DCT. Výsledek se přičítá k proměnné *sumResult* a konečný výsledek je uložen v matici *resultingInverseTransform*.

Funkce vypadá následovně:

```
function [resultingInverseTransform] = myidct(inputTransform)

% IDCT
resultingInverseTransform = inputTransform;

for x = 0:7
    for y = 0:7

        % for each pixel
        sumResult = 0;
        for u = 0:7
            for v = 0:7

                % Define  Cu, Cv
                if (u == 0)
                    Cu = sqrt(2) / 2;
                else
                    Cu = 1;
                end

                if (v == 0)
                    Cv = sqrt(2) / 2;
                else
                    Cv = 1;
                end

                sumResult = sumResult + (1/4) * Cu * Cv * inputTransform(u + 1, v + 1)*...
                    cos((2 * x + 1) * u * pi / 16) * cos((2 * y + 1) * v * pi / 16);
            end
        end

        % IDCT coefficients
        resultingInverseTransform(x + 1, y + 1) = sumResult;
    end
end
end
```

Následně byl rastr převeden z barevného prostoru YCbCr do RGB dle:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0091 & -0.0032 & 1.3955 \\ 1.0091 & -0.3472 & -0.7206 \\ 1.0091 & 1.7689 & -0.0066 \end{pmatrix} \begin{pmatrix} Y \\ C_B \\ C_R \end{pmatrix} - \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}.$$

Výsledný dekomprimovaný byl zobrazen.

```
% Output image

output_image = uint8(zeros(size(input_image)));
output_image(:, :, 1) = uint8(R_output);
output_image(:, :, 2) = uint8(G_output);
output_image(:, :, 3) = uint8(B_output);

% Display the output image
imshow(output_image);
```

VÝPOČET STŘEDNÍCH KVADRATICKÝCH CHYB:

Z barevných složek RGB před a po kompresi jsou vypočteny čtverce jejich rozdílů.

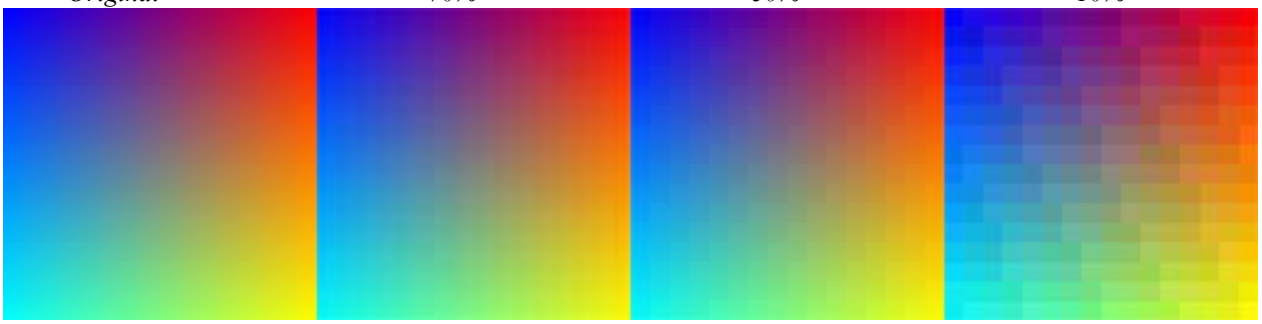
```
delta_R = R_output - R;
delta_G = G_output - G;
delta_B = B_output - B;

delta_R_squared = delta_R .* delta_R;
delta_G_squared = delta_G .* delta_G;
delta_B_squared = delta_B .* delta_B;
```

Z nich potom střední kvadratické chyby:

```
mean_R_error = sqrt(sum(sum(delta_R_squared)) / (m * n))
mean_G_error = sqrt(sum(sum(delta_G_squared)) / (m * n))
mean_B_error = sqrt(sum(sum(delta_B_squared)) / (m * n))
```

VÝSTUPY:

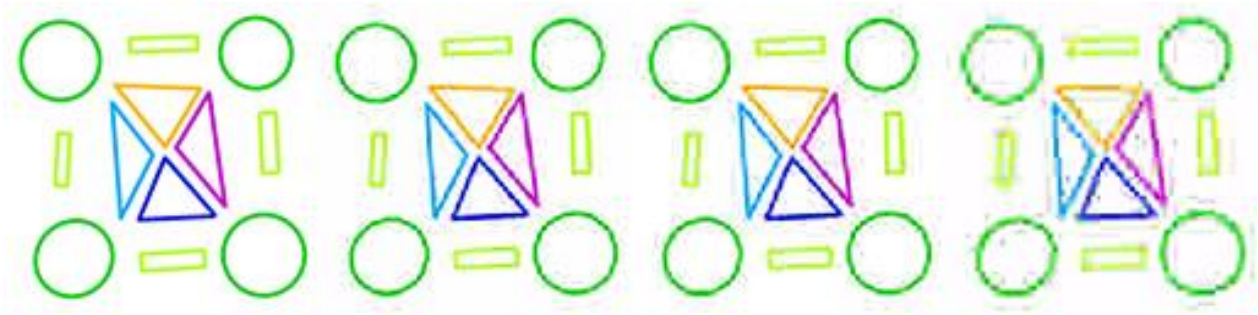
Originál *70%* *50%* *10%*



Originál *70%* *50%* *10%*

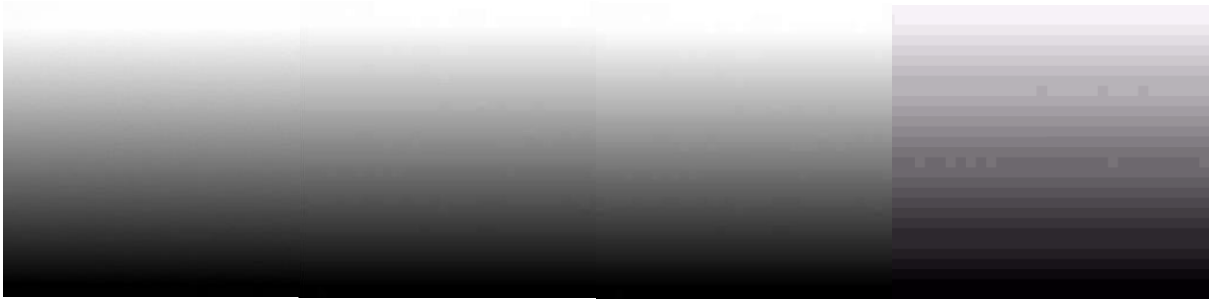
Originál *70%* *50%* *10%*

Originál *70%* *50%* *10%*



Originál *70%* *50%* *10%*

Originál *70%* *50%* *10%*



(Testované obrázky mají rozměry 256x256 pixelů a 8bitovou barevnou hloubku)

STŘEDNÍ KVADRATICKÉ ROZDÍLY RGB SLOŽEK U JEDNOTLIVÝCH RASTRŮ:

Barevný přechod			
q	m **R**	m **G**	m **B**
10	6.9176	5.8898	7.6127
50	1.3830	1.5802	1.4239
70	1.5584	1.2253	1.8471

Fotografie			
q	m **R**	m **G**	m **B**
10	24.8709	18.5183	18.4788
50	14.5617	11.9530	11.8101
70	12.3691	10.3868	10.8524

Barva			
q	m **R**	m **G**	m **B**
10	1.0518	0.7839	10.7503
50	1.0785	0.1448	1.6011
70	0.4614	0.6022	1.7733

Vektorová kresba			
q	m **R**	m **G**	m **B**
10	29.3531	21.2747	31.0191
50	14.6640	10.5718	15.2256
70	12.0683	8.5931	12.8919

Fotografie ve stupních šedi			
q	m **R**	m **G**	m **B**
10	11.8866	11.7771	12.0776
50	7.0236	7.0236	7.0236
70	6.1276	6.1054	6.1642

Škála ve stupních šedi			
q	m **R**	m **G**	m **B**
10	4.8865	4.3274	5.3701
50	1.0865	1.0865	1.0866
70	1.3656	1.2841	1.5185

ZÁVĚR:

Díky výše uvedených tabulek vidíme, že JPG komprese je nejvíce vhodná pro rastry o jedné barvě. Nejméně vhodná je pro rastry vektorové kresby, jelikož vektorový obrázek má ostré hrany s extrémními barevnými přechody, které se díky kompresi rozmažou. U barevných fotografií není vhodné volit faktor komprese nižší jak 70%-80%, jelikož potom obraz znatelně degraduje a ztrácí rozmanitost barev. Černobílé fotografie jsou na JPEG kompresi mnohem méně náchylné, jelikož jsou si původní pixely barevně blíže.

Pro ideální funkci je nezbytné správně zvolit faktor komprese, tak aby byl soubor po kompresi co nejmenší a zároveň bylo zachováno co nejvíce obrazových informací.

Dne 8.12.2023 v Praze

Josef Jehlička, Kateřina Chromá, Štěpán Šedivý