ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE FAKULTA STAVEBNÍ KATEDRA GEOMATIKY					
Název předmětu: **Geoinformatika**					
Úloha: U1	**Název úlohy:** JPEG komprese rastru				
Akademický rok: 2024/2025	**Semestr:** zimní	**Studijní skupina:** 102C	**Vypracoval:** Michal Kovář Filip Roučka	**Datum:** 22. 10. 2024	**Klasifikace:**

1 Zadání:

Implementujte algoritmus pro JPEG kompresi/dekompresi rastru v prostředí MATLAB (popř. v programovacím jazyce dle vlastního výběru), zahrnující tyto fáze:

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

a to **bez** využití vestavěných funkcí

Kompresní algoritmus otestujte na různých typech rastru: rastr v odstínech šedi, barevný rastr (viz tabulka) vhodného rozlišení a velikosti (max 128x128 pixelů) s různými hodnotami faktoru komprese $q = 10, 50, 70$
Pro každou variantu spočtěte střední kvadratickou odchylku m jednotlivých RGB složek:

$$m = \sqrt{\frac{\sum_{i=0}^{mn}(z - z')^2}{mn}}$$

Výsledky umístěte do přehledných tabulek pro jednotlivá q. Na základě výše vypočtených údajů zhodnoťte, ke kterým typům dat je JPEG komprese nejvíce a naopak nejméně vhodná.

2 Bonus:

- *Resamplování rastru některou z metod.*
- *Konverze pixelů do* **ZIG-ZAG** *sekvencí.*
- *Huffmanovo kódování.*
- *Náhrada* **DCT** *s využitím diskrétní Fourierovy transformace.*
- *Náhrada* **DCT** *s využitím diskrétní vlnkové transformace.*

3 Popis problému

Rastrová data jsou využívána velmi často protože na zobrazení dat s velké většiny nepotřebují specializovaný software, problém nastává u operacích spojených s uschováním, přenosem či editací. Tento problém se projevuje zejména u barevných rastrů s vysokým prostorovým rozlišením a barevnou hloubkou. Pomocí komprimačních algoritmů můžeme zefektivnit práci s rastrovými daty. Joint Photographic Experts Group neboli JPEG komprese je komprese statických a i pohyblivých rastrů. Je vhodná pro kompresi přirozených rastrů a není vhodná u rastrů vzniklých z vektorových rastrů. JPEG komprese funguje na faktu že lidské oko vnímá barvu pomocí čípků a tyčinek, přičemž tyčinky jsou citlivé na jas a čípky na barvu. V lidském oku je přibližně 170 milionů tyčinek a jenom 6-7 milionů, z toho vyplývá že lidské oko je vnímavější ke změně jasu, než ke změně barvy. Nevýznamné změny barev jsou odstraňovány, změny jasu jsou naopak s co největší přesností uchovány[1].
Jedním nejčastěji využívaných barevných modelů (RGB) pro nekomprimované rastrová data je založen na aditivním mícháním jednotlivých barevných složek (čím je větší sytost barevných složek, tím je výsledná barva světlejší). Model RGB však nevyhovuje pro JPEG kompresi model RGB je tak v prvním kroku převeden do modelu YC_BC_R. Popis barevných odstínů v tomto modelu je blízky skutečnému fyziologickému vnímání barev. Model je představován třemi složkami jednou jasovou Y a dvěma chrominančními C_B a C_R[1].
JPEG komprese se skládá z několika kroků[1]:

- Převod $(RGB) \longrightarrow (Y, C_B, C_R)$
- Downsamplování rastru
- Transformace pomocí diskrétní kosinové transformace
- Kvantizace pomocí kvantizační matice

- ZIG-ZAG sekvence
- Huffmanovo kódování

JPEG dekomprese se skládá z opačného postupu než při kompresi[1]:

- Huffmanovo kódování dekomprese
- Převod ZIG-ZAG sekvence na matice
- Dekvantizace pomocí kvantizační matice
- Transformace pomocí inverzní diskrétní kosinové transformace
- Upnsamplování rastru
- Převod $(Y, C_B, C_R) \longrightarrow (RGB)$

4 Popis metod:

4.1 Resamplování rastru

Pro resamplování rastru byla vytvořena třída `Resample`, která obsahuje metody pro downsampling a pro upsampling. Tyto metody zajišťují, že chrominanční komponenty obrazu jsou správně zmenšeny a následně zvětšeny, což je důležité pro efektivní kompresi a dekompresi. Pro tento účel byla vytvořena třída Resample s následujícími metodami:

- `MyDResampleNN` - Downsampluje matici pomocí metody nejbližšího souseda.
- `MyUResampleNN` - Upsampluje matici pomocí metody nejbližšího souseda.
- `MyDResample2X2` - Downsampluje matici pomocí kernelu [2x2], nový pixel vznikne jako průměr kernelu.
- `MyDResample3x3` - Downsampluje matici pomocí kernelu [3x3], nový pixel vznikne jako průměr kernelu.
- `MyUResample2X2` - Upsampluje matici pomocí kernelu [2x2], nové pixely vznikne pomocí lineární interpolace v kernelu.

4.2 Konverze pixelů do ZIG-ZAG sekvencí

Při kompresi byly převedeny bloky 8x8 pixelů do Zig-Zag sekvencí. Tím byla převedena 2D data na 1D. Pro tento účel byla vytvořena třída `ZigZag` s metodami `to` a `from`. Metoda `to` převádí matici do vektoru pomocí Zig-Zag sekvence a metoda `from` převádí vektor zpět do matice.

4.3 Huffmanovo kódování

Pro další snížení velikosti dat bylo použito Huffmanovo kódování. Tento algoritmus vytváří optimální kód pro každý symbol na základě jeho četnosti, což vede k efektivní kompresi dat. Pro tento účel byla vytvořena třída `MyHuffman` s následujícími metodami:

- `CipherHuff` - Komprimuje sekvenci pomocí Huffmanova kódování.
- `DecipherHuff` - Dekomprimuje sekvenci Huffmanova kódovaná.
- `WriteFiles` - Vytvoří složku s výsledky Huffmanova kódování.
- `ReadFiles` - Ze složky přečte a dekomprimuje data Huffmanovým Kódováním.

4.4 Transformace

Pro nejdůležitější krok celé komprese, který představuje převod mezi prostorovými souřadnicemi jednotlivých pixelů do prostorových frekvencí. Pro tento účel byla vytvořena třída Transformace s následujícími metodami:

- `mydct` - Na datech provede diskrétní kosinovou transformaci.
- `myidct` - Na datech provede inverzní diskrétní kosinovou transformaci.
- `mydwt` - Na datech provede diskrétní vlnkovou transformaci.
- `myidwt` - Na datech provede inverzní diskrétní vlnkovou transformaci.
- `myfft` - Na datech provede diskrétní Fourierovu transformaci.
- `myifft` - Na datech provede inverzní diskrétní Fourierovu transformaci.

5 Postup

5.1 Načtení a zobrazení obrázku

Obrázek je načten pomocí funkce `imread` a zobrazen pomocí `imshow`:

```
originalImage = imread('colour_2.bmp');
figure(1)
imshow(originalImage);
title('Uncompressed Image');
```

5.2 Volba parametrů komprese

Jsou nastaveny parametry komprese, jako je faktor komprese `q`, typ transformace `type_of_trans`, typ resamplingu `type_of_resample` a zda se má použít Huffmanovo kódování `use_huffman`:

```
q = 50;
type_of_trans = 'dwt';
type_of_resample = '2X2';
use_huffman = 'YES';
```

5.3 Transformace RGB do YCBCR modelu

RGB složky obrázku jsou převedeny do YCBCR modelu pomocí následujících rovnic[1][2]:

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

5.4 Downsampling chrominančních složek

Chrominanční složky (Cb a Cr) jsou downsamplovány pomocí zvolené metody (2X2 nebo NN):

```
CB_downsampled = feval(strcat('Resample.MyDResample', type_of_resample), CB);
CR_downsampled = feval(strcat('Resample.MyDResample', type_of_resample), CR);
```

5.5 Kvantizace koeficientů

Jsou definovány kvantizační matice pro Y a C složky a aktualizovány podle faktoru komprese `q`[1][2]:

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

$$Qc = \frac{50}{q} \cdot \begin{pmatrix} 17 & 18 & 24 & 47 & 66 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 69 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}$$

5.6 JPEG komprese a dekomprese

Provedení JPEG komprese a dekomprese zahrnuje transformaci, kvantizaci a případné Huffmanovo kódování:

```
[Y_zigzag] = compression(Y, Qy, type_of_trans);
[CB_zigzag] = compression(CB_downsampled, Qc, type_of_trans);
[CR_zigzag] = compression(CR_downsampled, Qc, type_of_trans);

if strcmp(use_huffman, 'YES')
    % Huffmanovo kódování
    % ...
end

[Y] = decompression(Y_zigzag, Qy, type_of_trans);
[Cb] = decompression(CB_zigzag, Qc, type_of_trans);
[Cr] = decompression(CR_zigzag, Qc, type_of_trans);
```

5.7 Upsampling Chrominančích složek

Chrominanční složky jsou upsamplovány zpět na původní velikost:

```
Cb = feval(strcat('Resample.MyUResample', type_of_resample), Cb);
Cr = feval(strcat('Resample.MyUResample', type_of_resample), Cr);
```

5.8 Transformace zpět do RGB

YCBCR složky jsou převedeny zpět do RGB formátu[1][2]:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0000 & 0.0000 & 1.4020 \\ 1.0000 & -0.3441 & -0.7141 \\ 1.0000 & 1.7720 & -0.0001 \end{pmatrix} \begin{pmatrix} Y \\ CB - 128 \\ CR - 128 \end{pmatrix}$$

```
Rd = Y + 1.4020*(Cr-128);
Gd = Y - 0.3441*(Cb-128) - 0.7141*(Cr-128);
Bd = Y + 1.7720 * (Cb-128) - 0.0001*(Cr-128);

Ri = uint8(Rd);
Gi = uint8(Gd);
```

```
Bi = uint8(Bd);

ras2(:,:,1) = Ri;
ras2(:,:,2) = Gi;
ras2(:,:,3) = Bi;

figure(2)
imshow(ras2);
title('Compressed Image');
```

5.9 Výpočet střední kvadratické odchylky

Střední kvadratická odchylka jednotlivých RGB složek je vypočtena podle následujícího vzorce:

$$m = \sqrt{\frac{\sum_{i=0}^{m \cdot n} (z_i - z_i')^2}{m \cdot n}}$$

V MATLABu je tento výpočet proveden následujícím způsobem:

```
dR = R - Rd;
dG = G - Gd;
dB = B - Bd;

dR2 = dR.^2;
dG2 = dG.^2;
dB2 = dB.^2;

sigR = sqrt(sum(sum(dR2))/(m*n));
sigG = sqrt(sum(sum(dG2))/(m*n));
sigB = sqrt(sum(sum(dB2))/(m*n));
```

Kde: - dR, dG, dB jsou rozdíly mezi původními a dekomprimovanými složkami. - $dR2, dG2, dB2$ jsou druhé mocniny těchto rozdílů. - $sigR, sigG, sigB$ jsou výsledné střední kvadratické odchylky pro jednotlivé složky.

Tímto způsobem je možné kvantifikovat rozdíly mezi původním a dekomprimovaným obrazem pro každou složku RGB.

5.10 Náhrada DCT s využitím diskrétní Fourierovy transformace (DFT)

Jako alternativa k diskrétní kosinové transformaci (DCT) byla implementována diskrétní Fourierova transformace (DFT). DFT byla implementováná pomocí algoritmu FFT.[3] Diskrétní Fourierova transformace (DFT) je definována jako:

$$X(k) = \sum_{n=0}^{N-1} x(n) \cdot e^{-j\frac{2\pi}{N}kn}$$

kde:

- $X(k)$ je k-tý koeficient ve frekvenční doméně,
- $x(n)$ je n-tý vzorek v prostorové doméně,
- N je počet vzorků,
- j je imaginární jednotka ($j = \sqrt{-1}$).

5.10.1 2D Fourierova transformace

Pro zpracování obrazu (2D matice) se využívá 2D DFT, která je definována jako:

$$X(u,v) = \sum_{x=0}^{M-1} \sum_{y=0}^{N-1} f(x,y) \cdot e^{-j2\pi\left(\frac{ux}{M} + \frac{vy}{N}\right)}$$

kde:

- $X(u, v)$ je koeficient ve frekvenční doméně,
- $f(x, y)$ je hodnota pixelu v prostorové doméně,
- M a N jsou rozměry obrazu,
- j je imaginární jednotka ($j = \sqrt{-1}$).

5.10.2 Rychlá Fourierova transformace (FFT)

Rychlá Fourierova transformace (FFT) je optimalizovaný algoritmus pro výpočet DFT, který výrazně snižuje počet potřebných operací. Konkrétně byl využit Cooley-Tukeyův algoritmus, který má časovou složitost $O(n \log n)$ na rozdíl od přímého výpočtu Fourierovy transformace, který má časovou náročnost $O(n^2)$. Cooley-Tukeyův algoritmu využívá rekurzi k rozdělení FFT na menší části.[4] Matematicky je FFT definována jako:

$$F(k) = \begin{cases} x(k) & \text{pro } n \leq 1 \\ \text{FFT}(x_{\text{sudý}}) + e^{-2i\pi(k-1)/n} \cdot \text{FFT}(x_{\text{lichý}}) & \text{pro } 1 \leq k \leq \frac{n}{2} \\ \text{FFT}(x_{\text{sudý}}) - e^{-2i\pi(k-1)/n} \cdot \text{FFT}(x_{\text{lichý}}) & \text{pro } \frac{n}{2} < k \leq n \end{cases}$$

kde:

- $x_{\text{sudý}}$ jsou sudé indexy vstupního vektoru x
- $x_{\text{lichý}}$ jsou liché indexy vstupního vektoru x

5.10.3 Použité proměnné

- $F(k)$: Výsledek FFT pro k-tou frekvenci.
- $x(n)$: Vstupní signál v časové (prostorové) doméně.
- n: Počet vzorků (délka vstupního signálu).
- k: Index frekvence, pro kterou se FFT počítá.
- j: Imaginární jednotka ($j = \sqrt{-1}$).

```
function F = fft(x)
    n = length(x);
    if n <= 1
        F = x;
    else
        even = fft(x(1:2:end));
        odd = fft(x(2:2:end));
        F = zeros(1, n);
        for k = 1:n/2
            t = exp(-2i * pi * (k-1) / n) * odd(k);
            F(k) = even(k) + t;
            F(k + n/2) = even(k) - t;
        end
    end
end
```

5.10.4 Dvourozměrná rychlá Fourierova transformace (FFT2)

Funkce `fft2` provádí dvourozměrnou Fourierovu transformaci na matici. Nejprve se provede FFT na každém řádku matice a poté na každém sloupci výsledné matice. Matematicky je dvourozměrná rychlá Fourierova transformace definována jako:

$$F(u, v) = \text{FFT2}(f(x, y)) = \text{FFT}\left(\text{FFT}(f(x, y))\right)$$

```
function F = fft2(X)
    [rows, cols] = size(X);
    F_complex = zeros(rows, cols);
    for i = 1:rows
        F_complex(i, :) = fft(X(i, :));
    end
    for i = 1:cols
        F_complex(:, i) = fft(F_complex(:, i));
    end
    F = F_complex;
end
```

5.10.5 Inverzní dvourozměrná Fourierova transformace (IFFT2)

Funkce `ifft2` provádí inverzní dvourozměrnou Fourierovu transformaci na matici. Nejprve se provede IFFT na každém sloupci matice a poté na každém řádku výsledné matice.

```
function x = ifft2(X)
    [rows, cols] = size(X);
    x_complex = zeros(rows, cols);
    for c = 1:cols
        x_complex(:, c) = ifft(X(:, c).');
    end
    for r = 1:rows
        x_complex(r, :) = ifft(x_complex(r, :));
    end
    x = x_complex;
end
```

5.10.6 Inverzní Fourierova transformace (IFFT)

Inverzní Fourierova transformace (IFFT) převádí obraz z frekvenční domény zpět do prostorové domény. Matematicky je definována jako:

$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) \cdot e^{j \frac{2\pi}{N} kn}$$

kde:

- $x(n)$ je n-tý vzorek v prostorové doméně,
- $X(k)$ je k-tý koeficient ve frekvenční doméně,
- N je počet vzorků,
- j je imaginární jednotka ($j = \sqrt{-1}$).

Algoritmus IFFT je podobný FFT, ale s opačným znaménkem v exponentu:

```
function x = ifft(X)
    n = length(X);
    if n <= 1
        x = X;
    else
        even = ifft(X(1:2:end));
        odd = ifft(X(2:2:end));
        x = zeros(1, n);
        for k = 1:n/2
            t = exp(2i * pi * (k-1) / n) * odd(k);
            x(k) = (even(k) + t) / 2;
            x(k + n/2) = (even(k) - t) / 2;
        end
    end
end
```

5.11 Náhrada DCT s využitím diskrétní vlnkové transformace (DWT)

Další alternativou k DCT je diskrétní vlnková transformace (DWT). DWT je schopna lépe zachytit detaily v obraze a může být výhodnější pro kompresi obrazů s vysokým rozlišením nebo složitými texturami.[5] Matematicky je DWT definována jako:

$$W_\psi[j,k] = \sum_{n=0}^{N-1} x[n]\psi_{j,k}[n]$$

kde:

- j je měřítko (scale),
- k je posun (translation),
- $\psi_{j,k}[n]$ je diskrétní vlnková funkce.

5.11.1 Použitý druh vlnky

V této implementaci byla použita Haarova vlnka, která je poměrně jednoduchá. Haarova vlnka je definována jako:

$$\psi(t) = \begin{cases} 1 & \text{pro } 0 \leq t < \frac{1}{2} \\ -1 & \text{pro } \frac{1}{2} \leq t < 1 \\ 0 & \text{jinak} \end{cases}$$

$$\phi(t) = \begin{cases} 1 & \text{pro } 0 \leq t < 1 \\ 0 & \text{jinak} \end{cases}$$

5.11.2 Dvourozměrná diskrétní vlnková transformace (DWT2)

Pro zpracování obrazu se využívá DWT2, která se provádí aplikací DWT na řádky a následně na sloupce obrazu. Výsledkem jsou čtyři sub-pásma: LL (nízkopásmové), LH (nízkopásmové horizontální), HL (nízkopásmové vertikální) a HH (vysokopásmové). Algoritmus 2D DWT pro zpracování obrazu je následující:

```
function output = dwt2(block)
    [rows, cols] = size(block);
    L = zeros(rows, cols/2);
    H = zeros(rows, cols/2);
    for i = 1:rows
        even_cols = block(i, 1:2:end);
        odd_cols = block(i, 2:2:end);
        L(i, :) = (even_cols + odd_cols) / 2;
        H(i, :) = (even_cols - odd_cols) / 2;
    end
    LL = zeros(rows/2, cols/2);
    LH = zeros(rows/2, cols/2);
    HL = zeros(rows/2, cols/2);
    HH = zeros(rows/2, cols/2);
    for j = 1:cols/2
        even_rows_L = L(1:2:end, j);
        odd_rows_L = L(2:2:end, j);
        even_rows_H = H(1:2:end, j);
        odd_rows_H = H(2:2:end, j);
        LL(:, j) = (even_rows_L + odd_rows_L) / 2;
        LH(:, j) = (even_rows_L - odd_rows_L) / 2;
        HL(:, j) = (even_rows_H + odd_rows_H) / 2;
        HH(:, j) = (even_rows_H - odd_rows_H) / 2;
    end
    output = [LL, LH; HL, HH];
end
```

5.11.3 Inverzní 2D DWT (IDWT2)

Inverzní diskrétní vlnková transformace (IDWT) obnovuje původní signál z jeho vlnkových koeficientů. Matematicky je IDWT definována jako:

$$x[n] = \sum_{j,k} W_\psi[j,k]\psi_{j,k}[n]$$

kde:

- $W_\psi[j,k]$ jsou vlnkové koeficienty,
- $\psi_{j,k}[n]$ jsou diskrétní vlnkové funkce.

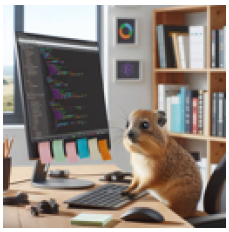
Algoritmus 2D IDWT pro zpracování obrazu je následující:

```
function block = idwt2([LL, LH; HL, HH])
    [rows, cols] = size(LL);
    LL_sub = LL(1:4, 1:4);
    LH_sub = LL(1:4, 5:8);
    HL_sub = LL(5:8, 1:4);
    HH_sub = LL(5:8, 5:8);
    L = zeros(rows, cols/2);
    H = zeros(rows, cols/2);
    for j = 1:cols/2
        even_rows_L = LL_sub(:, j) + LH_sub(:, j);
        odd_rows_L = LL_sub(:, j) - LH_sub(:, j);
        even_rows_H = HL_sub(:, j) + HH_sub(:, j);
        odd_rows_H = HL_sub(:, j) - HH_sub(:, j);
        L(1:2:end, j) = even_rows_L;
        L(2:2:end, j) = odd_rows_L;
        H(1:2:end, j) = even_rows_H;
        H(2:2:end, j) = odd_rows_H;
    end
    block = zeros(rows, cols);
    for i = 1:rows
        even_cols = L(i, :) + H(i, :);
        odd_cols = L(i, :) - H(i, :);
        block(i, 1:2:end) = even_cols;
        block(i, 2:2:end) = odd_cols;
    end
end
```

6 Vstupní data

Vstupními daty jsou obrázky ve formátu bitmapa (BMP), což je nekomprimovaný formát, který zajišťuje, že nedochází ke ztrátě kvality obrazu při ukládání. Obrázky jsou čtvercové s rozměry 128x128 pixelů. Konkrétně byly využité obrázky colour.bmp a greyscale.bmp nacházející se v adresáři images. Tyto obrázky byly vytvořeny pomocí nástroje DALL-E 3 na základě příkazu "damán, který programuje".

Nekomprimovaný barevný rastr



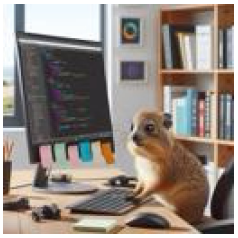
Nekomprimovaný šedotónový rastr

7 Výstupní data

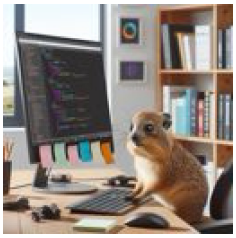
7.1 Komprimovaný a zpět dekomprimovaný barevný rastr

Barevný rastr: dct, 2X2, q=10

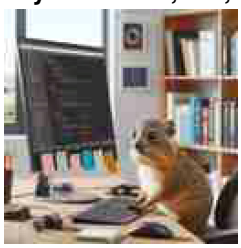
Barevný rastr: dct, 2X2, q=50



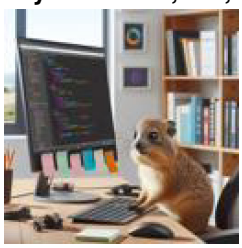
Barevný rastr: dct, 2X2, q=70



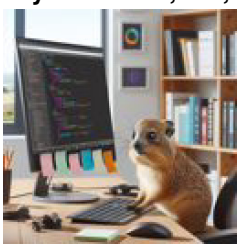
Barevný rastr: dct, NN, q=10



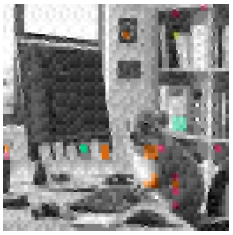
Barevný rastr: dct, NN, q=50



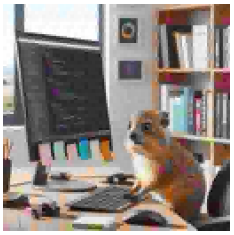
Barevný rastr: dct, NN, q=70



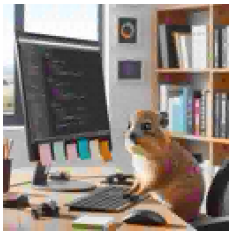
Barevný rastr: dwt, 2X2, q=10



Barevný rastr: dwt, 2X2, q=50

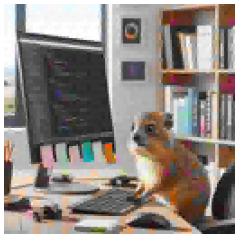


Barevný rastr: dwt, 2X2, q=70

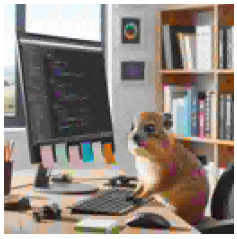


Barevný rastr: dwt, NN, q=10

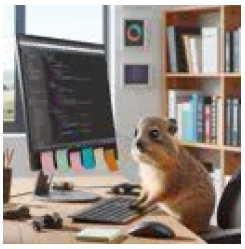
Barevný rastr: dwt, NN, q=50



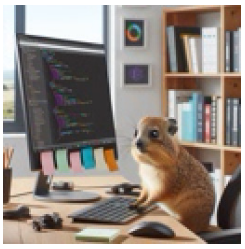
Barevný rastr: dwt, NN, q=70



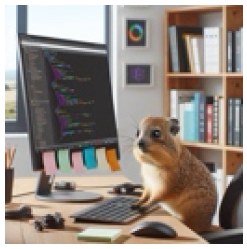
Barevný rastr: fft, 2X2, q=10



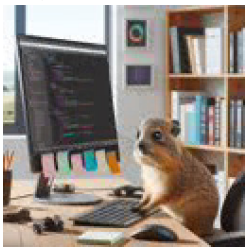
Barevný rastr: fft, 2X2, q=50



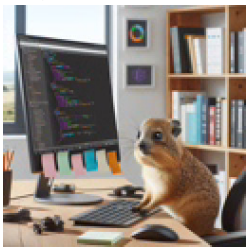
Barevný rastr: fft, 2X2, q=70



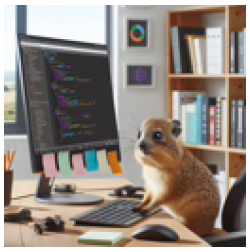
Barevný rastr: fft, NN, q=10



Barevný rastr: fft, NN, q=50



Barevný rastr: fft, NN, q=70



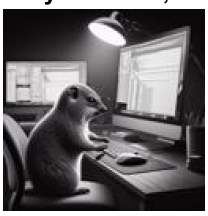
Tabulka 1. : Směrodatné odchylky RGB složek pro různé metody transformace, resamplování a kompresní faktory

Trans.	**Res.**	**q = 10**			**q = 50**			**q = 70**		
		σ_R	σ_G	σ_B	σ_R	σ_G	σ_B	σ_R	σ_G	σ_B
DCT	2X2	14.2314	11.5176	15.8831	8.7282	5.6081	9.6174	7.9765	4.8319	8.9358
DCT	NN	14.5588	11.5901	16.2380	9.3283	5.8469	10.3252	8.6363	5.0443	9.7421
DWT	2X2	26.3872	21.5015	28.8999	13.9211	9.8437	15.9432	11.9487	8.0544	13.5283
DWT	NN	26.9222	21.6850	28.8214	14.7302	9.9927	16.6923	12.6275	8.3376	14.3233
FFT	2X2	8.9595	5.7295	9.9961	5.7171	2.5141	6.3373	5.5416	2.3159	6.1004
FFT	NN	9.6429	5.9092	10.5269	6.9759	2.9553	7.7958	6.7097	2.7307	7.5133

7.2 Komprimovaný a zpět dekomprimovaný šedotonový rastr

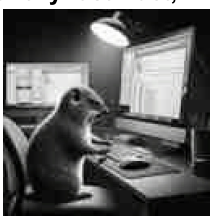
Šedotonový rastr: dct, 2X2, q=10

Šedotonový rastr: dct, 2X2, q=50

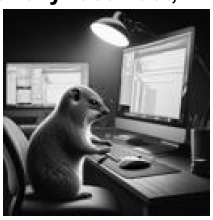


Šedotonový rastr: dct, 2X2, q=70

Šedotonový rastr: dct, NN, q=10

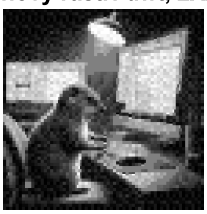


Šedotonový rastr: dct, NN, q=50



Šedotonový rastr: dct, NN, q=70

Šedotonový rastr: dwt, 2X2, q=10



Šedotonový rastr: dwt, 2X2, q=50

Šedotonový rastr: dwt, 2X2, q=70

Šedotonový rastr: dwt, NN, q=10

Šedotonový rastr: dwt, NN, q=50

Šedotonový rastr: dwt, NN, q=70

Šedotonový rastr: fft, 2X2, q=10

Šedotonový rastr: fft, 2X2, q=50

Šedotonový rastr: fft, 2X2, q=70

Šedotonový rastr: fft, NN, q=10

Šedotonový rastr: fft, NN, q=50

Šedotonový rastr: fft, NN, q=70

Tabulka 2: Směrodatné odchylky RGB složek pro různé metody transformace, resamplování a kompresní faktory

Trans.	**Res.**	**q = 10**			**q = 50**			**q = 70**		
		σ_R	σ_G	σ_B	σ_R	σ_G	σ_B	σ_R	σ_G	σ_B
DCT	2X2	9.9493	9.9081	9.9058	4.7716	4.7358	4.7866	3.8600	3.8242	3.8465
DCT	NN	10.0412	9.9457	9.9770	4.7513	4.7272	4.7839	3.8516	3.8229	3.8393
DWT	2X2	19.1238	19.0917	19.0951	8.0933	8.0272	8.0325	6.6940	6.6152	6.6196
DWT	NN	19.1797	19.1052	19.1421	8.2075	8.0723	8.1219	6.8302	6.6710	6.7260
FFT	2X2	4.5747	4.5585	4.5701	1.2294	1.1626	1.2278	0.9380	0.8555	0.9405
FFT	NN	4.5633	4.5571	4.5627	1.2030	1.1579	1.1896	0.9061	0.8477	0.8870

8 Závěr

Na základě výše uvedených tabulek lze vidět, že různé metody komprese a resamplování mají odlišný vliv na kvalitu obrazu po dekompresi.

8.1 Transformace:

- **DCT**: Je vhodná pro různé typy obrazů, ale při nižších kompresních faktorech (např. q = 10) dochází k výraznějšímu zhoršení kvality. Při vyšších kompresních faktorech (q = 50 a q = 70) je kvalita obrazu lepší, ale stále dochází k určité ztrátě detailů.
- **DWT**: Tato metoda vykazuje vyšší směrodatné odchylky, což naznačuje větší ztrátu kvality než DCT. Je možné že se někde v během výpočtu vyskytla chyba, jelikož podle [6] by měla DWT poskytovat lepší výsledky než DCT.
- **FFT**: Tato metoda dosahuje nejnižších směrodatných odchylek, což znamená, že nejméně poškozuje komprimovaný rastr. Je vhodná pro obrazy, kde je důležité zachovat co nejvíce detailů. Nevýhodou je, že při kompresi se pracuje s komplexními čísly, které dále komplikují Huffmanovo kódování a představují větší objem dat.

8.2 Resamplování:

- **2X2**: Tento typ resamplování obecně vykazuje nižší směrodatné odchylky než NN, což znamená, že méně poškozuje výsledný obraz.
- **NN (Nearest Neighbor)**: Tento typ resamplování má tendenci k vyšším směrodatným odchylkám, což naznačuje větší ztrátu kvality.

Doporučení:

- Pro barevné fotografie je vhodné volit kompresní faktor alespoň 50-70
- Pro černobílé fotografie je JPEG komprese méně náchylná k degradaci kvality, protože původní pixely jsou si barevně blíže.
- Pro vektorové kresby a obrazy s ostrými hranami je JPEG komprese méně vhodná, protože může dojít k rozmazání ostrých přechodů.

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

8.3 Další možné neřešené problémy a náměty na vylepšení

- **Optimalizace Huffmanova kódování**: Při použití FFT komprese se pracuje s komplexními čísly, což může komplikovat Huffmanovo kódování. Bylo by vhodné prozkoumat možnosti optimalizace tohoto procesu. Momentálně je vektor komplexních čísel rozdělen pomocí podmínky na komplexní a reálnou část a každá část je kódována samostatně.
- **Adaptivní komprese**: Jako vylepšení by bylo možné přidat adaptivní metody komprese, které by se dynamicky přizpůsoboval obsahu obrazu a optimalizovaly kompresi pro konkrétní rastr.
- **Zachování barevné věrnosti**: Bylo by možné využít také metody, které by lépe zachovávaly barevnou věrnost obrazu po kompresi, zejména u obrazů s jemnými barevnými přechody.
- **Vyšší rozlišení**: Bylo by dále možné upravit kód tak, aby lépe komprimoval i rastry s větším rozlišením.
- **Univerzalita**: Optimalizace procesu tak, aby byl schopen zpracovávat nejen čtvercová data, ale také data s rozměry, které nejsou násobky osmi.
- **Upsampeling**: Při upsamplingu s využitím kernelu [2x2] aplikovat pokročilejší metodu než lineární interpolaci.

Odkazy

[1] Tomáš Bayer. *Algoritmy pro kompresi rastrových dat.* URL: https://web.natur.cuni.cz/~bayertom/images/courses/Apk/komprese.pdf (cit. 21. 10. 2024).

[2] ČVUT Katedra geoinformatiky. *Geoinformatika 2 - Přednášky.* URL: https://github.com/k155cvut/ygei/blob/main/prednasky/geoinf2.pdf (cit. 21. 10. 2024).

[3] Wikipedia. *Fast Fourier transform.* URL: https://en.wikipedia.org/wiki/Fast_Fourier_transform (cit. 21. 10. 2024).

[4] Wikipedia. *Cooley–Tukey FFT algorithm.* URL: https://en.wikipedia.org/wiki/Cooley%E2%80%93Tukey_FFT_algorithm (cit. 12. 11. 2024).

[5] Wikipedia. *Wavelet transform.* URL: https://en.wikipedia.org/wiki/Wavelet_transform (cit. 21. 10. 2024).

[6] Wikipedia. *JPEG 2000.* URL: https://en.wikipedia.org/wiki/JPEG_2000 (cit. 21. 10. 2024).