ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

FAKULTA STAVEBNÍ, OBOR GEODÉZIE A KARTOGRAFIE
KATEDRA GEOMATIKY

název předmětu

GEOINFORMATIKA

číslo úlohy	*název úlohy*		
1	JPEG komprese rastru		
školní rok	*zpracovali*	*datum*	*klasifikace*
2025/26	Králič Adam, Matějková Barbora	19. 11. 2025	

Technická zpráva

1 Zadání úlohy

Úkolem bylo implementovat JPEG kompresi rastru ve zvoleném programovacím jazyku (Matlab), která by zahrnovala tyto fáze:

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

a to bez vystavěných funkcí použitého programu.

Výstupem je i testování kompresního algoritmu na jednotlivých typech rastrových souborů s různými hodnotami faktoru komprese $q = 10, 50, 70$, včetně vypočtení střední kvadratické odchylky m jednotlivých RGB složek.

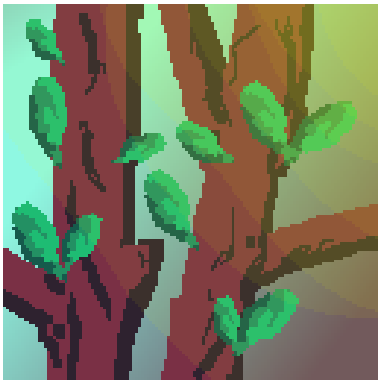
$$m = \sqrt{\frac{\sum_{i=0}^{m \cdot n} (z - z')^2}{m \cdot n}}$$

Možností bylo implementovat další kroky či jiné kompresní algoritmy (viz tab. 1), které byly všechny v této úloze implementovány.

Krok	**Hodnocení**
JPG komprese/dekomprese rastru.	20b
Resamplování rastru.	+5b
Konverze prvků do ZIG-ZAG sekvencí.	+10b
Huffmanovo kódování.	+15b
Náhrada DCT s využitím DFT.	+15b
Náhrada DCT s využitím DWT.	+15b
Max celkem:	**80b**

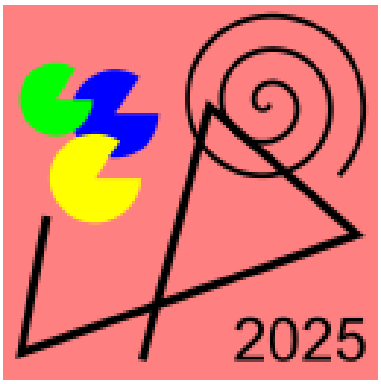
Tabulka 1: Kroky a jejich hodnocení

Pro testování kompresí jednotlivými transformacemi byly vybrány tři obrázky – grafika typu pixel art s barevnými přechody o rozměrech 128×128 px, jež byla vytvořena v programu Aseprite, fotografie 256×256 px a vektorová grafika 128×128 px s liniovými prvky z programu Inkscape.



(a) barevný obrázek

(b) fotografie



(c) vektorová grafika

Obrázek 1: Použité obrázky k testování kompresí

2 Postup výpočtu

2.1 Komprese

1. **Nahrání souboru:** soubur je nahrán pomocí funkce `imread(nazSouboru.bmp)`, následně je rozdělen na 3 barevné složky RGB, kde každá má stejný rozměr $m \times n$.

2. **Převod `RGB` na `YUV` model**: `YUV` model také obsahuje 3 složky – jasovou Y a dvě barevné (chromatické) složky C_B a C_R.

   ```
   % RGB to 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;
   ```

3. **Přepočet intervalu:** interval byl převeden z hodnot {0, 255} na {-255, 255}.

   ```
   Y = 2*Y - 255;
   Cb = 2*Cb - 255;
   Cr = 2*Cr - 255;
   ```

4. **Převzorkování rastru:** Rastr je převzorkován pomocí kernelu o libovolné kladné celočíselné hodnotě (např. 2×2). Výsledkem je rastr o stejné velikosti, avšak obsahující zprůměrované hodnoty pro dané vyhledávací okno.

5. **Rozdělení do submatic 8×8:** Pro JPEG kompresi je zásadní rozdělení rastru do 8×8 submatic, u kterých probíhá komprese samostatně (což je pro rastr po dekompresi pozorovatelné jako nespojitost mezi sousedními bloky).

6. **Transformace:** Stále bezztrátový krok komprese. Jedná se o převedení opakujících se hodnot pomocí funkčního vztahu do menšího počtu hodnot. Ke každé transformaci existuje i inverzní zápis, kterým lze zpět dostat originální data.

   V této práci byly použity direktivní kosinová `DCT`, direktivní Fourierova `DFT` a direktivní vlnková `DWT` transformace.

7. **Kvantizace koeficientů:** Jedná se o nejvíce ztrátovou část JPEG komprese. Cílem je vypuštění koeficientů s malou hodnotou.

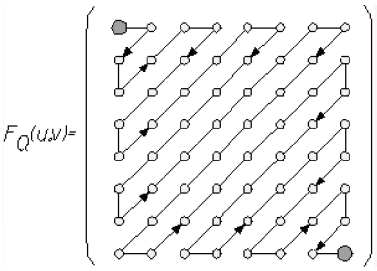
   Základní postup: kvantizace provádí dělení matice $F(u, v)$ po prvcích kvantizační maticí $Q(u, v)$. Kvantizační matice mohou být různé pro jasovou i chromatickou složku. Ta se následně vydělí faktorem komprese q, který ve vysoké míře určuje ztrátu komprese a provede se celočíselné zaokrouhlení.

   Ukázka jasové kvantizační matice:

```
Qy = [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];
```

8. **„Poskládání“ do nové matice:** Jednotlivé submatice 8×8 jsou poskládány do nové matice o rozměrech původních matice na své patřičné pozice.

9. **Zig-Zag sekvence:** Matice je dále přestrukturována do vektoru o velikosti $m \cdot n \times 1$. Je přeskládán po diagonální Zig-Zag sekvenci podle obr. 2.



Obrázek 2: Zig-Zag sekvence

10. **Huffmanova kódování:** Posledním krokem komprese je převedení hodnot do bitové podoby podle četnosti unikátních hodnot. Tuto podobu je možné získat právě Huffmanovo kódováním.

2.2 Dekomprese

Dekomprese se od komprese liší pouze v převrácení výpočetních kroků a použití inverzních transformací a jiných potřebných výpočtů.

Jednotlivé kroky: dekomprese Huffmanova kódu → inverzní Zig-Zag sekvence → rozdělení do submatic 8×8 → dekvantizace → inverzní transformace → poskládání do nové matice → inverzní přepočet hodnot zpět na interval $\{0, 255\}$ → převod `YUV` do `RGB` modelu → vytvoření nového souboru.

2.3 Vyhodnocení

Pro **každou barevnou složku**, použitý faktor komprese q, použitý rastr a zvolené velikosti převzorkování byly vypočteny střední kvadratické odchylky m.

Ukázka kódu:

```
dR = R - R_new;
dG = G - G_new;
dB = B - B_new;

sigmaR = sqrt(sum(sum(dR.^2))/(m*n));
sigmaG = sqrt(sum(sum(dG.^2))/(m*n));
sigmaB = sqrt(sum(sum(dB.^2))/(m*n));
```

3 Vytvořené funkce

3.1 Direktivní kosinová transformace

Povinná funkce, která byla vytvořena částečně na cvičení. V rámci úlohy byla dále doplněna její inverzní varianta. Výpočet probíhal pro submatice 8×8, které byly následně skládány do původního rastru.

Rovnice `DCT`:

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

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

Inverzní kosinová transformace, společně s Fourierovo a vlnkovou transformací, transformuje obraz zpátky do původní podoby[1].

Rovnice `IDCT`:

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

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

Realizace direktivní kosinové transformace v programu Matlab:

```
function [img_t] = dct(img)       % discrete cosine transformation
img_t = img;
for u = 0:7                       % process lines
    if (u == 0)                   % compute cu
        cu = sqrt(2)/2;
    else
        cu = 1;
```

[1] kdyby neproběhla kvantizace, výsledek by byl identický

```
    end
    for v = 0:7                    % process columns
        if (v == 0)                % compute cv
            cv = sqrt(2)/2;
        else
            cv = 1;
        end
        fuv = 0;                  % compute sum
        for x = 0:7               % process lines
            for y = 0:7           % process columns
                fuv = fuv + 1/4 * cu *cv * img(x+1, y+1) * ...
                    cos((2*x+1)*u*pi/16)*cos((2*y+1)*v*pi/16);
            end
        end
        img_t(u+1, v+1) = fuv;    % update raster
    end
end
```

DCT převádí obrazový signál na posloupnost signálů, které se odlišují amplitudou a frekvencí, tvořící frekvenční spektrum.

3.2 Direktivní Fourierova transformace

Výpočet i princip metody DFT je dosti podobný DCT. Slouží k převedení signálu do frekvenční složky.

Výsledkem DFT je matice o velikosti původního rastru, jejíž hodnoty obsahují i imaginární složku; s ní však nebylo dále pracováno. Výpočet probíhal obdobně pro submatice 8×8.

Rovnice DFT:

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

Rovnice IDFT:

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

Zápis v programu Matlab je dosti podobný zápisu DCT, liší se pouze matematický zápis.

```
function [img_t] = dft(img)              % discrete fourier transformation
[m, n] = size(img);
clear j;                                 % to make sure 'j' is a complex unit
img_t = img;
for u = 0:7                              % process lines
    for v = 0:7                          % process columns
        fuv = 0;                         % compute sum
        for x = 0:7                      % process lines
            for y = 0:7                  % process columns
                fuv = fuv + img(x+1, y+1)*exp(-j*2*pi*((u*x/m) + (v*y/n)));
            end
```

```
        end
        img_t(u+1, v+1) = fuv;            % update raster
    end
end
```

3.3 Direktivní vlnková (Wavelet) transformace

DWT rozděluje původní data do 4 frekvenčních pásem: LL (2× použitý nízkofrekvenční filtr), LH, HL (kombinace nízkofrekvenčního a vysokofrekvenčního filtru) a HH (2× použitý vysokofrekvenční filtr).

Na rozdíl od metod DCT a DFT byla tato metoda použita pro celý rastr o velikosti 128×128 pixelů. Tato metoda je i oddělena ve svém vlastním skriptu komprese_dwt.m, jelikož postup výpočetních kroků je jiný.

Pro rastrová data, která mají spíše spojitý průběh, má největší význam LL, jenž se následně může opět rozdělit do 4 pásem, a výpočet lze provádět iteračně dále, dokud lze rastr ještě rozdělit.

V této úloze byla použita Haarova vlnka, což je nejstarší a nejjednodušší vlnka. Výpočet probíhal nejdřív podle sloupců, kde se vždy braly hodnoty ze dvou sousedních sloupců, nízkofrekvenční filtr (L) tyto hodnoty sčítal a vysokofrekvenční filtr (H) odčítal a výsledné hodnoty byly uloženy do jedné buňky → 2× méně sloupců. Obdobně probíhal výpočet podle řádek pro oba filtry L a H → 2× méně řádek. Výsledkem jsou 4 pásma (LL, LH, HL, HH), ze kterých bylo pro následnou dekompozici použito pouze pásmo LL.

Inverzní DWT byla vypočtena z dekomprimovaného LL a původních složek LH, HL, HH, ze kterých byl vytvořen rastr o původní velikosti.

Ukázka výpočtu pro DWT

```
function [LL, LH, HL, HH] = dwt2d(img)
    % processes columns (n columns -> n/2)
    [m, n] = size(img);
    L = zeros(m, n/2);
    H = zeros(m, n/2);
    for i = 1:m               % creates low pass and high pass filter
        k = 1;
        for j = 1:2:n
            L(i, k) = (img(i,j) + img(i,j+1)) / sqrt(2);   % lowpass
            H(i, k) = (img(i,j) - img(i,j+1)) / sqrt(2);   % highpass
            k = k + 1;
        end
    end
    % processes rows and creates LL, LH, HL and HH filters
    LL = zeros(m/2, n/2);
    LH = zeros(m/2, n/2);
    HL = zeros(m/2, n/2);
    HH = zeros(m/2, n/2);
    for j = 1:n/2
        k = 1;
        for i = 1:2:m
            LL(k,j) = (L(i,j) + L(i+1,j)) / sqrt(2);       % lowpass rows
            LH(k,j) = (L(i,j) - L(i+1,j)) / sqrt(2);
            HL(k,j) = (H(i,j) + H(i+1,j)) / sqrt(2);       % highpass rows
```

```
            HH(k,j) = (H(i,j) - H(i+1,j)) / sqrt(2);
            k = k + 1;
        end
    end
end
```

3.4 Huffmanovo kódování

Huffmanovo kódování se často používá jako poslední část před uložením komprimovaného souboru. Jde o převedení hodnot do bitové podoby, kde nejčastější hodnoty mají nejkratší bitový zápis a nejdelší bitový zápis má dvojce nejméně se vyskytujících hodnot.

Pro tyto hodnoty je potřeba mít uložený „slovník“ neboli soubor s převody mezi hodnotou a jejím bitovým zápisem.

Princip algoritmu: Data se přetvoří na seznam bodů, ze kterých se následně vyberou 2 nejméně četné hodnoty a vytvoří z nich uzly (levý uzel přijímá hodnotu 0, pravý 1). Z těchto hodnot vznikne další uzel o stupeň výše, jehož četnost je součet četností všech jeho synů a zařadí se do původního seznamu bodů, první dva body se ze seznamu vyřadí. Dále se opět vyberou dva body s nejmenší četností (v tomto případě to může být nově vzniklý uzel + nepoužitý uzel nebo 2 nepoužité uzly) a proces se opakuje, dokud seznam bodů neobsahuje ani jeden bod. Poslední uzel (kořen) by měl obsahovat četnost 100%.

```
% builds the huffman tree
while size(huffman_nodes,1) > 1                  % while there is more than 1 node
    % sorting to have the two with the lowest propability at the top
    huffman_nodes = sortrows(huffman_nodes, 2);

    left_node = huffman_nodes(1, :);             % takes the smallest node
    right_node = huffman_nodes(2, :);            % takes the second smallest node

    new_symbol = {left_node{1}, right_node{1}};  % store children
    new_prob = left_node{2} + right_node{2};
    new_node = {new_symbol, new_prob};           % creates a new node

    huffman_nodes(1:2, :) = [];                  % removes old nodes
    huffman_nodes(end+1, :) = new_node;          % appends a new node
end
```

Algoritmus v této úloze vrací novou matici s již zakódovanými bitovými hodnotami a slovník mezi původními hodnotami a jejich bitovým zápisem.

Slovník se vytváří zpětným průchodem proměnné `huffman_nodes`, kde každá buňka obsahuje dvě předchozí buňky, ze kterých se sama skládá. Levé buňce se při průchodu přidává prefix `'0'` a pravé `'1'`. Pro každý případ se takto vytvoří unikátní kód a je následně je uložen do slovníku.

```
    queue = {huffman_nodes{1,1}, ''};      % defining map for better variable storing
    codes = containers.Map('KeyType','double','ValueType','char');
    while length(queue) > 0                      % creates huffman codes
        node = queue{1,1};                       % pops the first element
        prefix = queue{1,2};
```

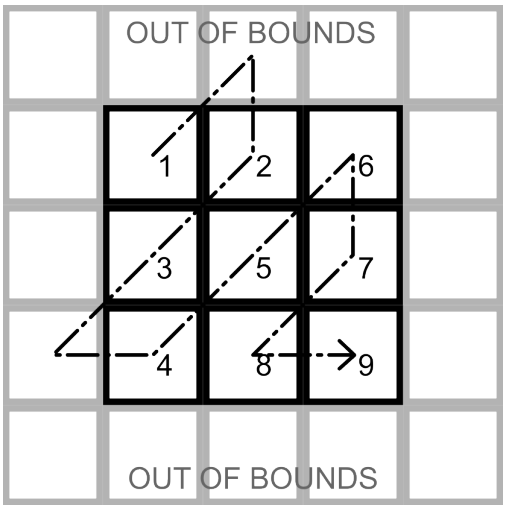
```
        queue(1,:) = [];                          % dequeue
        if iscell(node)                           % distributes prefixes to branches
            queue(end+1,:) = {node{1}, strcat(prefix,'0')}; % left br. appends '0'
            queue(end+1,:) = {node{2}, strcat(prefix,'1')}; % right br. appends '1'
        else
            if iscell(node)
                node = node{1};
            end
            codes(node) = prefix;                 % stores code in Map
        end
    end
```

Inverzní funkce přijímá vzniklý slovník a zakódovanou matici a vrací původní nezakódovanou matici.

3.5 ZIG-ZAG sekvence

Tento algoritmus slouží k vytvoření posloupnosti, kterou lze efektivně komprimovat → prvky se stejnými hodnotami budou za sebou.

Implementována je následovně: postupně se naplňuje nově vznikající vektor. Ukazatel se pohybuje po původní matici buďto nahoru doprava nebo dolu doleva. Vždy se provede test, zda by se ukazatel v příští iteraci dostal *out of bounds*. Pokud ano, změnil by se směr pohybu a ukazatel by se posunul na adekvátní pozici. Algoritmus se provádí tak dlouho, dokud se vektor celý „nenaplní".



Obrázek 3: chod ukazatele při zig-zag algoritmu

```
veta = false;           % rule veta, if one of the conditions for 'out_of_bounds' is
                        % true, the next iteration has to be valid
up = true;              % defines movement direction of pointer
while array_i <= m*n                      % until the array is not filled
    if up                                 % goes up right
        array(array_i) = image(i, j);
        array_i = array_i + 1;
        if (j>=n) & ~veta                 % if column index would be out of bounds
            up = false;                   % changes direction
```

```
            i = i + 1;
            veta = true;
        elseif (i<=1) & ~veta           % if row index would be out of bounds
            up = false;                 % changes direction
            j = j + 1;
            veta = true;
        else                            % normal situation - not out of bounds
            i = i - 1;
            j = j + 1;
            veta = false;
        end
    else                                % goes left down
        array(array_i) = image(i, j);
        array_i = array_i + 1;
        if (i>= m) & ~veta              % if row index would be out of bounds
            up = true;                  % changes direction
            j = j + 1;
            veta = true;
        elseif (j <= 1) & ~veta         % if column index would be out of bounds
            up = true;                  % changes direction
            i = i + 1;
            veta = true;
        else
            j = j - 1;                  % normal situation - not out of bounds
            i = i + 1;
            veta = false;
        end
    end
end
```

Inverzní funkce funguje na stejném principu, jenom se místo vektoru $(1,\ m \cdot n)$ naplňuje matice o původní velikosti $(m,\ n)$

3.6 Převzorkování (Resampling)

Jedná se změnu velikosti rastru na požadovanou velikost. Časté využití je například v mapové algebře či při komprimacích rastrových souborů.

V této úloze byly implementovány 2 funkce pro převzorkování a inverzní převzorkování. Vstupem je rastr a krok („level“) převzorkování.

Byla zde zvolena metoda vyhledávacího okna (*kernel*), které pro zvolený celočíselný krok (pro krok 2: kernel = (2,2)) byla uložena průměrná hodnota buněk v kernelu do nového rastru. Pokud byl tedy zvolen krok 2, výsledná velikost souboru je $2\times$ menší.

Pro následující výpočty byla každopádně vytvořena i inverzní funkce, která s tím samým krokem vytvoří nový rastr, kde naopak vezme hodnotu z předchozího rastru a vloží ji do buněk o velikosti původního vyhledávacího okna.

4 Výsledky

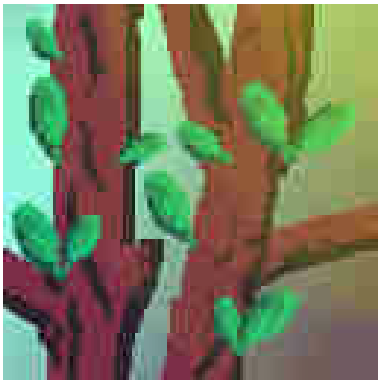
V rámci úlohy byly vyhotoveny všechny bonusové části, výpočetní skripty k nim jsou dostupné na githubu skupiny. Jedná se o dva hlavní skripty – jeden obsahující kompresi funkcemi DCT a DFT a druhý s DWT – a pomocné funkce vyhotovené povětšinou ve vlastních skriptech.

4.1 Barevný obrázek

Pro první obrázek (barevný pixel art) byly kromě jednotlivých transformací testovány i dva typy převzorkování, s hodnotou 1 a 2.

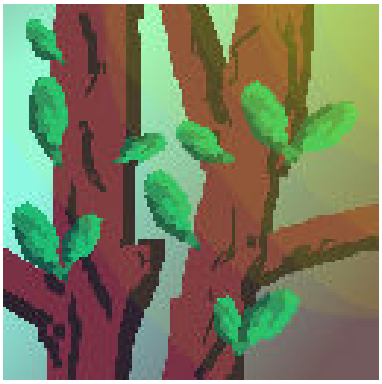
DCT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	15.781	13.291	11.343
1×1	50	10.281	7.235	6.056
1×1	70	8.769	5.962	5.177
2×2	10	16.006	16.610	13.929
2×2	50	14.205	16.779	13.506
2×2	70	13.985	16.865	13.473
DFT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	19.371	23.781	18.693
1×1	50	18.222	23.392	18.345
1×1	70	18.168	23.377	18.309
2×2	10	21.164	26.635	20.999
2×2	50	20.674	26.497	20.840
2×2	70	20.674	26.503	20.838
DWT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	84.818	67.455	84.122
1×1	50	21.608	13.793	22.576
1×1	70	14.188	10.071	18.878
2×2	10	88.957	75.198	88.186
2×2	50	34.440	35.984	34.783
2×2	70	30.340	34.728	32.504

Tabulka 2: Odchylky transformací pro barevný obrázek

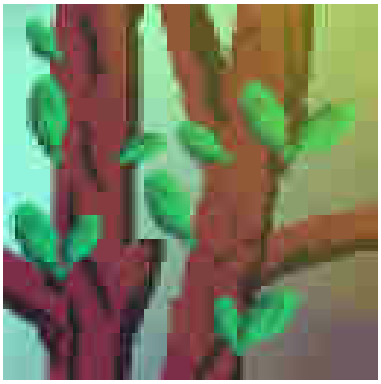


(a) $q = 10$

(b) $q = 50$



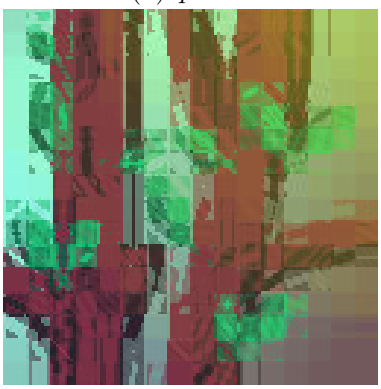
(c) $q = 70$



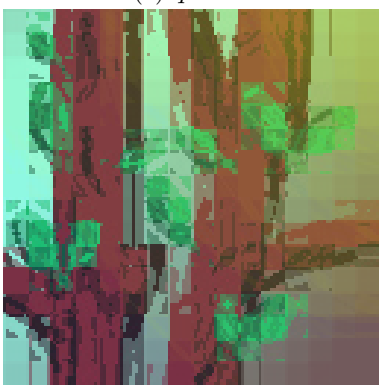
(d) $q = 10$

(e) $q = 50$

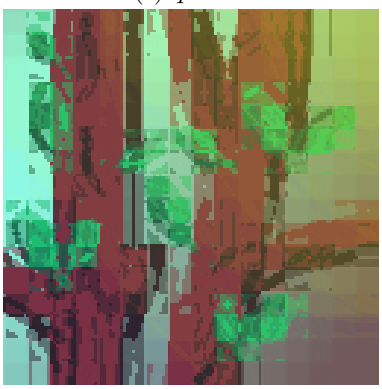
(f) $q = 70$



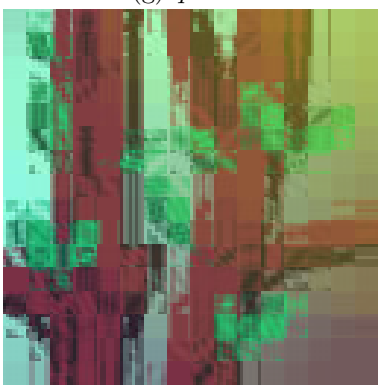
(g) $q = 10$



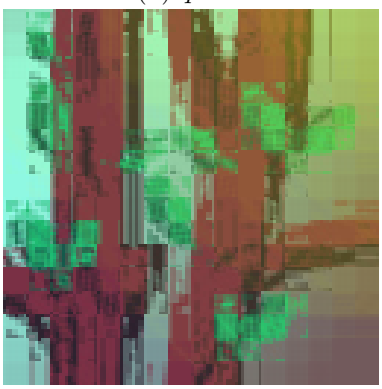
(h) $q = 50$



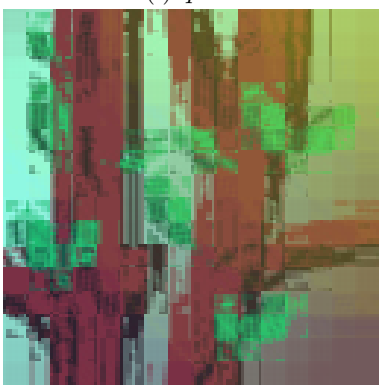
(i) $q = 70$



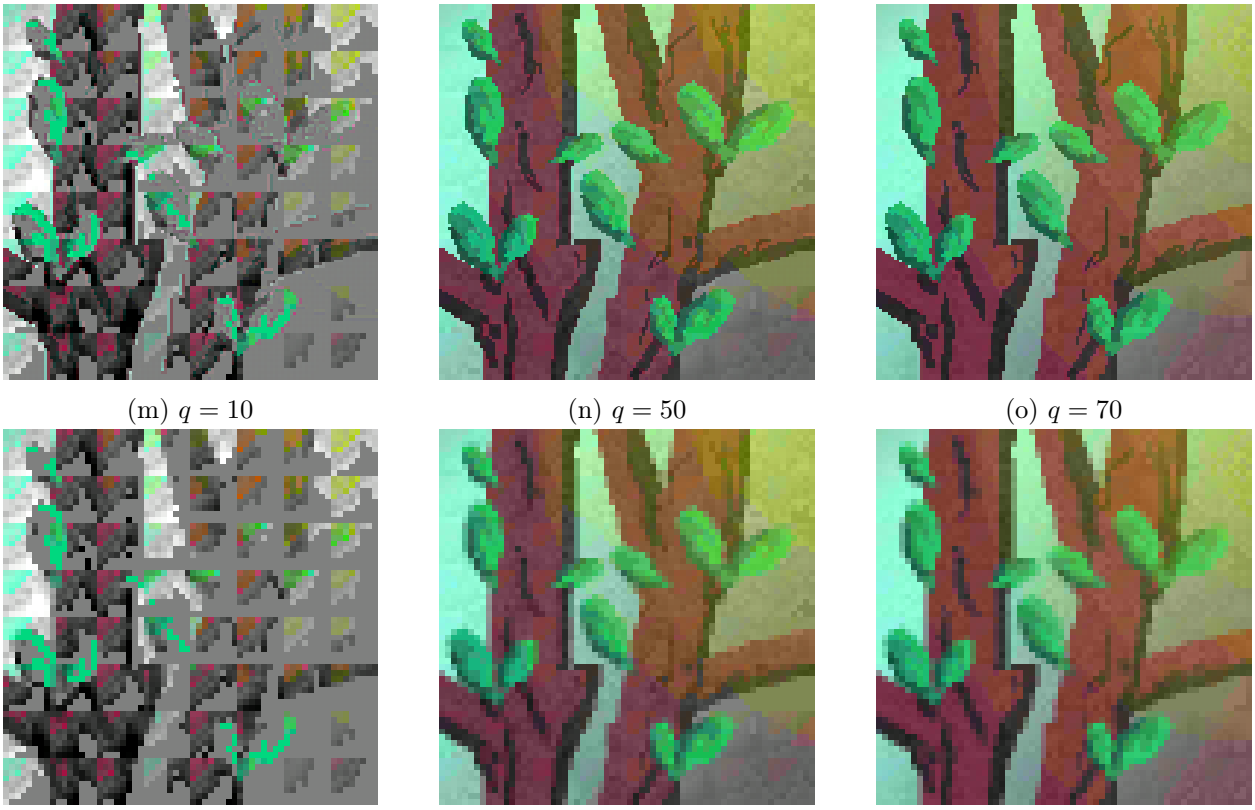
(j) $q = 10$



(k) $q = 50$



(l) $q = 70$



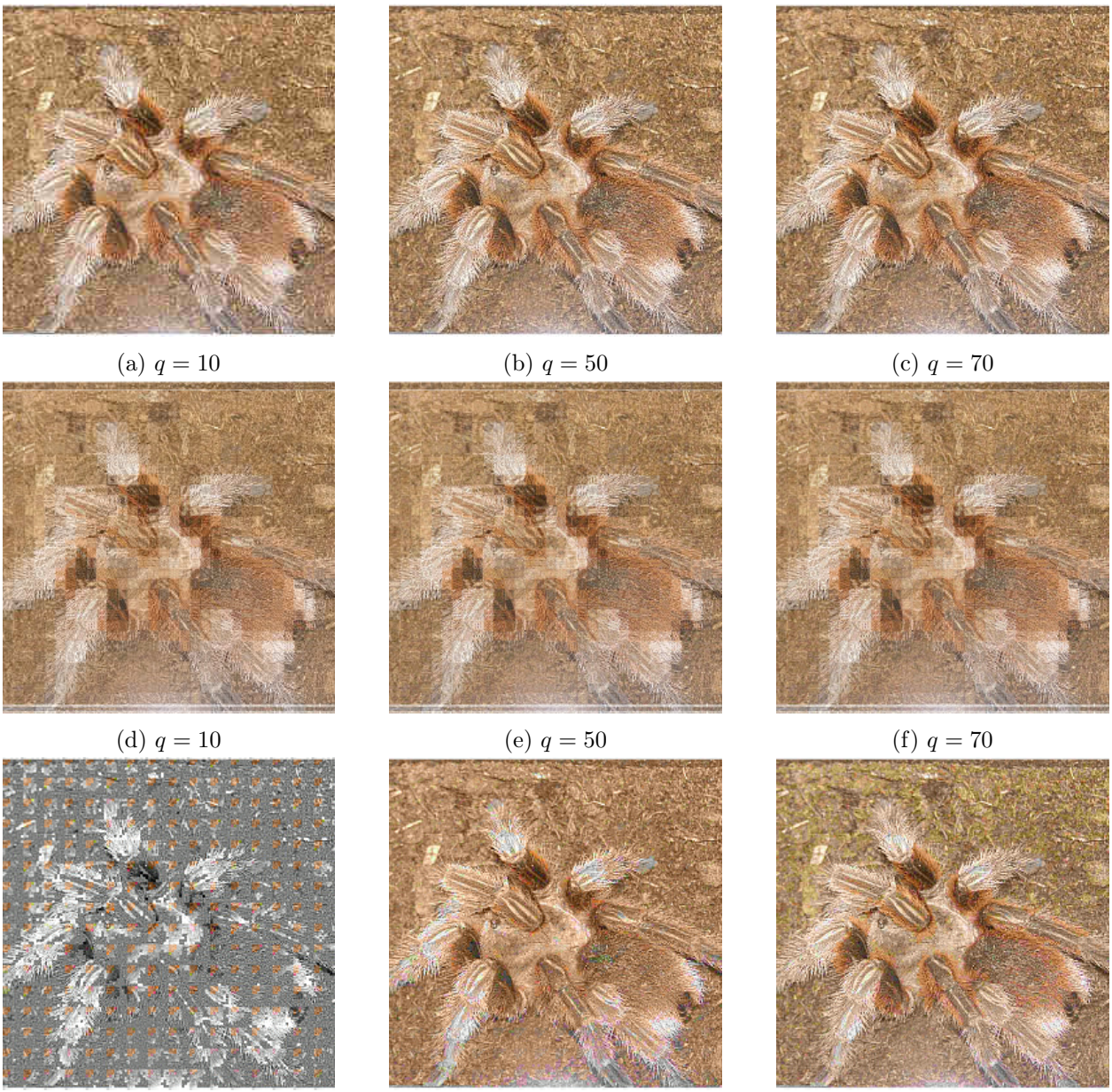
(m) $q = 10$ (n) $q = 50$ (o) $q = 70$

(p) $q = 10$ (q) $q = 50$ (r) $q = 70$

Obrázek 4: Barevný pixel art obrázek, (a) – (f) DCT, (g) – (l) DFT, (m) – (r) DWT, liché řádky resampling 1×1, sudé 2×2

DCT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	20.083	19.898	20.904
1×1	50	9.405	9.065	9.790
1×1	70	7.212	6.846	7.600
DFT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	21.852	22.572	23.876
1×1	50	21.316	22.138	23.310
1×1	70	21.292	22.125	23.280
DWT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	76.721	42.814	76.896
1×1	50	18.761	12.200	21.962
1×1	70	13.951	9.590	19.132

Tabulka 3: Odchylky transformací pro fotografii



(a) $q = 10$ (b) $q = 50$ (c) $q = 70$

(d) $q = 10$ (e) $q = 50$ (f) $q = 70$

(g) $q = 10$ (h) $q = 50$ (i) $q = 70$

Obrázek 5: Fotografie, a, b, c DCT, d, e, f DFT, g, h, i DWT

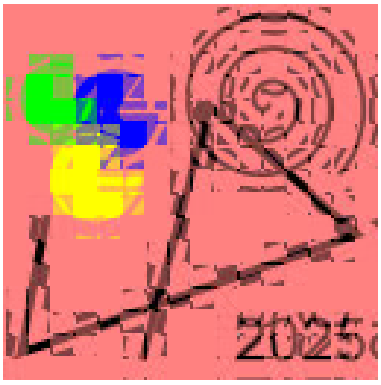
DCT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	21.134	12.723	16.068
1×1	50	10.020	6.360	8.387
1×1	70	7.919	5.158	6.951
DFT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	51.542	28.769	29.043
1×1	50	51.082	28.439	28.505
1×1	70	51.068	28.426	28.471
DWT				
resampling	q	σ_R	σ_G	σ_B
1×1	10	120.884	111.411	69.860
1×1	50	25.533	15.938	20.363
1×1	70	15.771	12.449	12.740

Tabulka 4: Odchylky transformací pro vektorovou grafiku

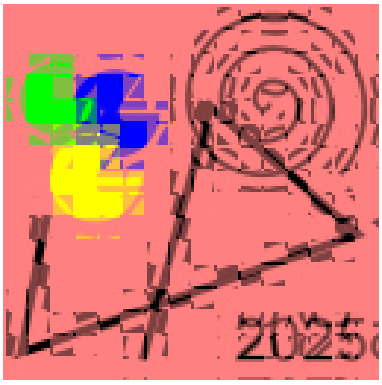
(a) $q = 10$

(b) $q = 50$

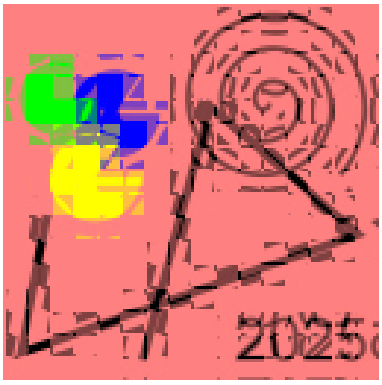
(c) $q = 70$



(d) $q = 10$

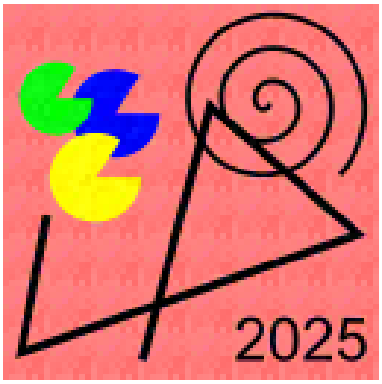


(e) $q = 50$



(f) $q = 70$

(g) $q = 10$



(h) $q = 50$

(i) $q = 70$

Obrázek 6: Vektorová grafika, a, b, c DCT, d, e, f DFT, g, h, i DWT

Závěr

Směrodatné odchylky jednotlivých barev samozřejmě závisí na hodnotě q, kdy nižší hodnoty obrazová data silněji poznamenaly, avšak lze vypozorovat jisté trendy i u jednotlivých transformací.

Po kompresi a následné dekompresi nejpodobněji originálu dopadly všechny obrázky při použití metody `DCT`. Odchylky jsou v tomto případě menší i při použití $q = 10$, než u ostatních metod při použití mírnějšího faktoru komprese $q = 70$.

Při `DFT` došlo k nápadnému rozložení obrazu na pixely obsahující velké množství reliktů. Lze zde pozorovat jednotlivé „vlny“, které jsou odlišné pro každý 8×8 blok. Nejmarkantnější je tento jev v případě liniových prvků u třetího obrázku, kde už tvoří překážku pro vnímání objektů. Odchylky se se zvyšujícím se q moc nesnižují, zapříčiňuje to nejspíš samotná vlastnost `DFT`, která nalezne pár vyskytujících se frekvencí v hledaném 8×8 bloku, není tedy nejvhodnější metodou pro použitá rastrová data, jelikož se nejedná o data s harmonickou vlastností.

`DWT` barvy rozkládala úplně, nejvíce nepřekvapivě v případě použité fotografie. Může za to nejspíše jiná metoda výpočtu, která spočívá v transformaci celého rastru, odkud se následně chyby ze ztráty komprese projeví ve všech 8×8 blocích obdobně.

Přílohy

- Příloha 1 - výpočetní skripty z programu Matlab

Reference

[1] Výklad ze cvičení – doc. Ing. Tomáš Bayer, Ph.D.

[2] 155YGEI Geoinformatika. Online. 2025. Dostupné z: https://geo.fsv.cvut.cz/gwiki/155YGEI_Geoinformatika. [cit. 2025-10-13]..

Podepsáno dne 19. 11. 2025 v Praze Králič Adam, Matějková Barbora