VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ

FACULTY OF MECHANICAL ENGINEERING

ÚSTAV MATEMATIKY

INSTITUTE OF MATHEMATICS

ALTERNATIVNÍ JPEG DEKODÉR

ALTERNATIVE JPEG IMAGE DECODER

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE — **Bc. JIŘÍ BUREŠ**

AUTHOR

VEDOUCÍ PRÁCE — **prof. Mgr. PAVEL RAJMIC, Ph.D.**

SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání diplomové práce

Ústav:	Ústav matematiky
Student:	**Bc. Jiří Bureš**
Studijní program:	Aplikované vědy v inženýrství
Studijní obor:	Matematické inženýrství
Vedoucí práce:	**prof. Mgr. Pavel Rajmic, Ph.D.**
Akademický rok:	2021/22

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma diplomové práce:

Alternativní JPEG dekodér

Stručná charakteristika problematiky úkolu:

Obrazový kodek JPEG je všeobecně znám a široce používán. Kompresní algoritmus pracuje na základě chytrého „useknutí“ harmonického rozvoje dvojrozměrných bloků pixelů. Tento postup však ústí ve známý artefakt (vlastnost) JPEG a tím jsou nevhodné úkazy poblíž ostrých hran v obraze po kompresi. Náhlé změny v obrazu, což odpovídá ostrým hranám, totiž z principu není možné vyjádřit neúplným harmonickým rozvojem v kmitočtové oblasti.
Práce se bude zabývat vytvořením nového kodeku (tj. kodéru, ale zde zejména dekodéru), který bude vycházet ze standardu JPEG, nicméně bude potlačovat výše zmíněné nežádoucí jevy pomocí dopnění a využití informace o hodnotách pixelů poblíž hran.

Cíle diplomové práce:

– Student se seznámí se zdroji týkajícími se klasického kodeku JPEG a nastuduje detailně existující kodér v MATLABu.
– Student pochopí důvod, proč typické nechtěné artefakty poblíž hran v obraze vznikají.
– Dále se student seznámí s tzv. řídkými reprezentacemi signálů a metodami konvexní optimalizace.
– Tyto vědomosti poté student využije k návrhu kodéru a zejména dekodéru obrazu. Kodér bude využívat hranových detektorů.
– Dekodér využívající proximálních optimalizačních algoritmů student naprogramuje v MATLABu.
– Poslední fází bude testování a vyhodnocení účinnosti nového kodeku v porovnání s „klasickým“ JPEG. – Srovnávání proběhne jak z hlediska kvality (minimálně kritéria PSNR, SSIM), tak rovněž rychlosti výpočtů.

Seznam doporučené literatury:

ELAD, M., Sparse and Redundant Representations: From Theory to Applications in Signal and Image Processing. New York: Springer, 2010.

RAJMIC, P., DAŇKOVÁ, M., Úvod do řídkých reprezentací signálů a komprimovaného snímání. Skriptum, Vysoké učení technické v Brně, 2014. ISBN 978-80-214-5169-8.

GONZALES, R., WOODS, R. Digital Image Processing. Prentice Hall, 3rd ed. ISBN 978-0131687288.

ŠOREL, M., BARTOŠ, M., Fast Bayesian JPEG Decompression and Denoising With Tight Frame Priors. IEEE Transactions on Image Processing vol. 26, 1, 2017.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2021/22

V Brně, dne

L. S.

prof. RNDr. Josef Šlapal, CSc. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

Abstrakt
Tato diplomová práce se zabývá obrazovým kodekem JPEG, detekcí hran v obraze, řídkými reprezentacemi signálů a proximálními algoritmy. Nejprve je popsáno fungování kodéru a dekodéru JPEG a teorie, ze které vychází. Poté je na základě teoretických poznatků sestaven nový proximální algoritmus a implementován do stávajícího algoritmu JPEG s cílem odstranit blokové relikty v dekédovaném obraze. Programová stránka je řešená v prostředí Matlab. Výsledky jsou zhodnoceny využitím metod MSE, PSNR a SSIM.

Summary
This thesis deals with the JPEG image codec, edge detection in images, sparse signal representations and proximal algorithms. First, the operation of the JPEG encoder and decoder and the theory underlying it are described. Then, based on the theoretical knowledge, a new proximal algorithm is constructed and implemented in an existing JPEG algorithm in order to remove block relics in the decoded image. The programming side is solved in Matlab environment. The results are evaluated using MSE, PSNR and SSIM methods.

Klíčová slova
zpracování obrazu, JPEG, detekce hran, řídkost, řídké reprezentace, proximální operátor, proximální algoritmy

Keywords
image processing, JPEG, edge detection, sparsity, sparse representation, proximity operator, proximal algorithms

BUREŠ, J.*Alternativní JPEG dekodér*. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2023. 86 s. Vedoucí prof. Mgr. Pavel Rajmic, Ph.D.

Prohlašuji, že jsem diplomovou práci *Alternativní JPEG dekodér* vypracoval samostatně pod vedením prof. Mgr. Pavla Rajmice, Ph.D. s použitím materiálů uvedených v seznamu literatury.

Bc. Jiří Bureš

Chtěl bych poděkovat... Chtěl bych poděkovat především vedoucímu diplomové práce prof. Mgr. Pavlu Rajmicovi, Ph.D. za jeho trpělivost, vlídný přístup při konzultacích a cenné rady.

Bc. Jiří Bureš

Obsah

Úvod

V této práci se budu zabývat kompresním algoritmem JPEG, který je v dnešní době jedním z nejrozšířenějších vůbec. JPEG je rychlý a má vynikající kompresní výsledky. Bohužel však je jeho komprese ztrátová, což má za následek nechtěné jevy v rekonstruovaném obraze. Tato práce si klade za cíl upravit původní JPEG algoritmus, tak aby tyto jevy byly potlačeny, v ideálním případě zcela odstraněny.

K úpravě algoritmu využijeme řídkých reprezentací signálu a proximálních optimalizačních algoritmů. Pokusíme se na základě teoretických podkladů sestavit vlastní proximální algoritmus, který s využitím předem uložených hran v obraze bude schopen rekonstruovat obraz tak, aby se co nejvíce blížil kvalitou svému vzoru.

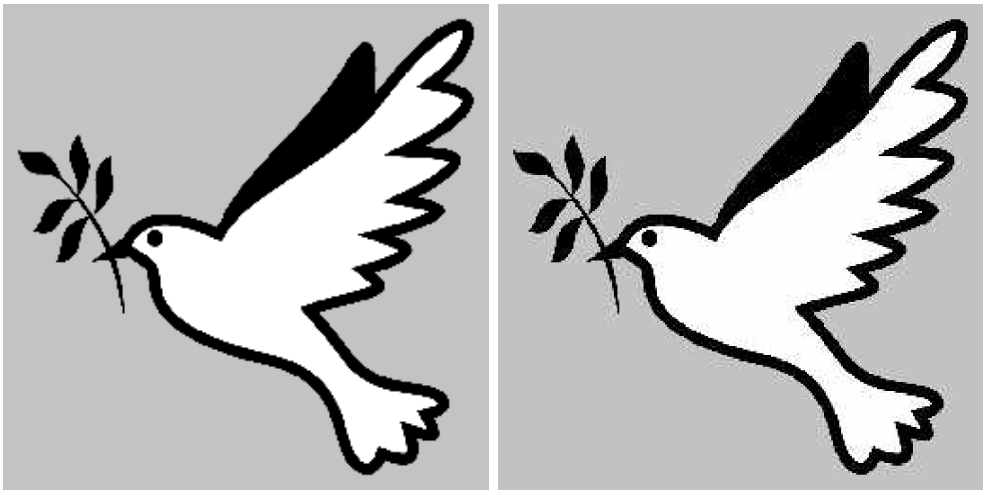
Kvalitu našeho algoritmu budeme testovat jak na sadě standardizovaných obrazů volně dostupných na internetu, tak i některých specificky vybraných obrazech. Hodnocení kvality proběhne na základě výsledných hodnot MSE, PSNR a SSIM.

1. Popis problému a nástin řešení

Jednou z typických vlastností komprese JPEG jsou obrazové artefakty objevující se zejména v blízkosti ostrých hran v obraze.

Dochází k nim vlivem ztráty vysokofrekvenčních složek při kvantizaci, kdy odstraňujeme jisté množství transformačních koeficientů. Přicházíme tak i tu část signálu, která se podílí na vykreslování detailů v obraze. To vede při rekonstrukci transformačního bloku k odchýlení od původního signálu a blok se vykreslí nesprávně.

Dobře patrné je to při porovnání následujících 2 obrázků. Obrázky vlevo jsou původní, obrázky vpravo byly zakódované a opět rekonstruované algoritmem JPEG.



(a) Orginální obrázek. (b) Obrázek po kompresi.

(c) Orginální obrázek. (d) Obrázek po kompresi.

Obrázek 1.1: Porovnání obrazů před a po JPEG kompresi. (Obrázek (a) získán z [70])

Návrh řešení tohoto problému spočívá v uložení dodatečné informace v podobě zvoleného procenta hranových bodů v originálním obrazu. Ty budou následně použité jako dodatečné vstupy do proximálního algoritmu založeného na obecném algoritmu navrženém v [12]. Algoritmus bude řešit úlohu konvexní optimalizace s cílem znovu vytvořit ztracené koeficienty.

Výsledkem by měl být obraz vizuálně lepší kvality než obraz získaný klasickým JPEG dekódováním a především by neměl obsahovat zmíněné relikty na hranách.

K odvození algoritmu bude využito teorie uvedené v této práci na základě odborných zdrojů. Samotný algoritmus bude realizován v prostředí MATLAB s využitím kódů klasického JPEG algoritmu, poskytnutých vedoucím práce prof. P. Rajmicem, viz [72].

2. Kompresní algoritmy

Ztrátové komprese obrazu využívají nadbytku informací v obraze, které není lidské oko schopné zachytit. Pokud tedy nevyužíváme obraz pro různé obrazové analýzy, můžeme dobře zvolenou část informace zcela vypustit. Rozdíl bude pro lidské oko nerozeznatelný nebo jen velmi nepatrný. Zároveň tak docílíme redukce výpočetní náročnosti a zmenšíme nároky na paměť při uchovávání obrazů. Tedy redukujeme kvantitu dat bez ztráty jejich kvality.

Existují způsoby pro měření jak kvantity, tak i kvality. V případě kvantity je to kompresní poměr „$x : 1$“ mezi datovou náročností komprimovaného a výchozího snímku. Lze jej vyjádřit v bitech na pixel (bpp). Pokud uvažujeme původní obraz v odstínech šedi (tj. 256 úrovní jasu), pak bude mít 8 bitů na pixel ($2^8 = 256$), tj 8bpp.

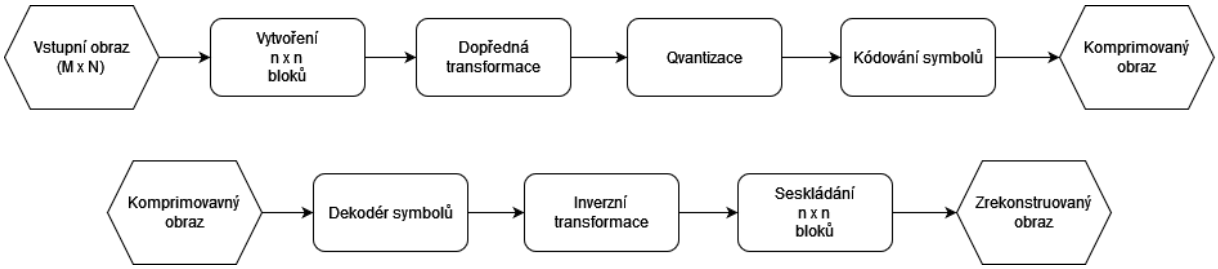
Pro hodnocení kvality využíváme kritéria jako MSE, PSNR (resp. SNR) či SSIM a blíže se jim budeme věnovat v kapitole 11.[56]

2.1. Blokové transformační kódování

Jak název napovídá, základem transformačního kódování je výběr vhodné (diskrétní) transformace, kterou aplikujeme na obraz (resp. na bloky, na které byl „rozsekán“). Vlastnosti takové transformace musí být: (bi)ortogonalita, reverzibilita, separovatelnost a rychlost. Aby ale byly ztrátové formáty opravdu účinné je nutné také použít dobře zvolenou kvantizaci a neztrátové metody kódování v závěrečné fázi.[56]

Kódovací algoritmus (kodér) nejprve provede dekompozici obrazu do bloků o stejné velikosti. Každý blok je následně zpracován samostatně za použití nějaké 2D lineární transformace (např. Fourierova transformace). Ta namapuje blok na množinu transformačních koeficientů, které jsou následně kvantizovány a zakódovány. Dekodér implementuje opačné pořadí kroků (s výjimkou kvantizační funkce) vzhledem ke kodéru, což dělá tyto algoritmy velmi přímočaré a relativně nenáročné na implementaci.

Schéma obvyklého algoritmu blokové transformace je na obrázku 2.1.



Obrázek 2.1: Schéma komprese a rekonstrukce obrazu.

Obvykle platí, že kodéry jsou složitější, dekodéry bývají jednodušší a rychlejší. Jak ale uvidíme později, v této práci toto pravidlo porušíme.

2.1.1. Výběr velikosti bloku

Vstupní obraz o rozměrech $M \times N$ je obvykle rozdělen na bloky o velikosti $n \times n$, které jsou následně transformovány a vytváří MN/n^2 polí transformovaných koeficientů, každé o velikosti $n \times n$.

Rozdělení zmenšuje míru korelace v obraze a zrychluje výpočet transformace obrazu. Pro velikost bloku je obvykle voleno celé číslo v mocninách 2. Nejčastěji používané velikosti jsou 8×8 a 16×16 pixelů. Obecně platí, že s větším blokem se zvyšuje jak úroveň komprese tak i výpočetní náročnost.[16][38]

2.1.2. Výběr transformace

Výběr vhodné 2D diskrétní transformace závisí na účelu použití a požadovaných vlastnostech jako třeba výpočetní náročnost. V [16] jsou obecná formulace v vlastnosti používaných transformací popsány následovně. Uvažujeme-li blok $g(x, y)$ o velikosti $n \times n$, pak jeho přímá/dopředná diskrétní transformace $T(u, v)$ může být obecně vyjádřena pomocí vztahu

$$T(u,v) = \sum_{x=0}^{n-1}\sum_{y=0}^{n-1} g(x,y)r(x,y,u,v) \tag{2.1}$$

pro $u, v = 0, 1, 2, \ldots, n-1$. Podobně pro daná $T(u, v)$, $g(x, y)$ může být vyjádřena zobecněná inverzní diskrétní transformace

$$T(u,v) = \sum_{x=0}^{n-1}\sum_{y=0}^{n-1} g(x,y)r(x,y,u,v) \tag{2.2}$$

pro $x, y = 0, 1, 2, \ldots, n-1$. V těchto rovnicích jsou $r(x, y, u, v)$ a $s(x, y, u, v)$ nazývány *přímá* (dopředná) a *inverzní* (zpětná) *transformační jádra* (tranformation kernels). Často jsou označovány také jako *bázové funkce* nebo bázové obrazy. Hodnoty $T(u, v)$ pro $u, v = 0, 1, 2, \ldots, n-1$ jsou *transformační koeficienty.*

Jádro je separabilní, pokud

$$r(x,y,u,v) = r_1(x,u) \cdot r_2(y,v). \tag{2.3}$$

Navíc je jádro symetrické, pokud je r_1 funkčně ekvivalentní r_2. V takovém případě rovnice (2.3) může být vyjádřena ve formě

$$r(x,y,u,v) = r_1(x,u) \cdot r_1(y,v). \tag{2.4}$$

Identické vlastnosti platí také pro inverzní jádro, pokud $r(x, y, u, v)$ nahradíme $s(x, y, u, v)$. Uvedené vlastnosti umožňují počítat 2D transformace jako odpovídající 1D transformace po řádcích a sloupcích.

Dopředné a zpětné transformační jádro ve výrazech (2.1) a (2.2) určuje typ transformace.

Výpočetně jednoduchá a v transformačním kódování také užitečná a používaná je Walsh-Hadamardova transformace (WHT, Walsh-Hadamard transform). Pro tuto transformaci mají transformační jádra podobu

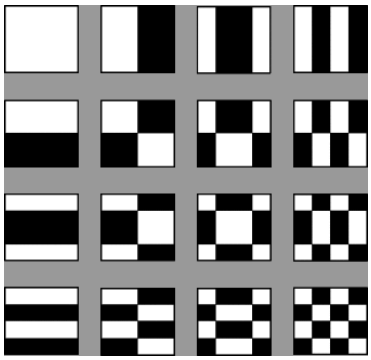
$$r(x,y,u,v) = s(x,y,u,v) = \frac{1}{n}(-1)^{\sum_{i=0}^{m-1} |b_i(x)p_i(u)+b_i(y)p_i(v)|}, \tag{2.5}$$

kde $n = 2^m$. Sumace, která se vyskytuje v exponentu, je počítána v modulo 2 aritmetice a $b_k(z)$ je k-tý bit (z prava do leva) v binární reprezentaci z. Například pokud je $m = 3$ a $z = 6$ (binárně 110), pak $b_0(z) = 0, b_1(z) = 1$ a $b_2(z) = 1$. Hodnoty $p_i(u)$ jsou počítány následovně

$$\begin{aligned} p_0(u) &= b_{m-1}(u) \\ p_1(u) &= b_{m-1}(u) + b_{m-2}(u) \\ &\vdots \\ p_{m-1}(u) &= b_1(u) + b_0(u), \end{aligned}$$

kde jsou opět všechny sumace prováděny v modulo 2. Podobně pro $p_i(v)$.

Na rozdíl například od jader DFT, které jsou tvořeny součty sinů a kosinů, se WHT skládá ze střídajících se hodnot 1 a -1. Ty tak vytváří jakoby šachovnicový vzor, viz obrázek 2.2. Každý blok se skládá ze $4 \times 4 = 16$ prvků (čtverců). Bílé značí +1 a černé -1. Přičemž pro obdržení konkrétního bloku udávají souřadnice $u, v = 0, 1, 2, 3$ pozici bloků a souřadnice $x, y = 0, 1, 2, 3$ udávají pozici čtverečku v bloku. Takže například $r(0, 1, 3, 3)$ je bílý čtvereček v první řadě, druhém sloupci úplně posledního bloku vpravo dole.



Obrázek 2.2: Bázové funkce WHT.

2.1.3. Kvantizace

Kvantizace (resp. kvantování) je ztrátový a nevratný proces. V oblasti ztrátové komprese obrazů kvantizací rozumíme techniku, při níž jsou vstupní vzorky reprezentovány se sníženou přesností. Pokud jsou vzorky zdroje kvantizovány jednotlivě, mluvíme o skalární kvantizaci; pokud jsou kvantizovány celé bloky vzorků společně, jedná se o vektorovou kvantizaci. V případech, kde nám jde o dosaženou kompresi, podává vektrorová kvantizace ve srovnání se skalární lepší výkon.[65]

Kvantizace v našem případě zahrnuje jak kvantizaci samotnou, tak i prahování. Přístup staví na myšlence, že největší koeficienty (tj. koeficienty reprezentující nejnižší frekvence) také nesou největší množství informace. Záměrem je tedy nulovat („zahodit“) ty koeficienty, které jsou dostatečně malé (tj. reprezentují vysoké frekvence), protože mají minimální dopad na kvalitu zrekonstruovaného obrazu, a zároveň dojde k úspoře při kódování do binární reprezentace.

Hodnotu konkrétního kvantizačního koeficientu $\hat{T}(u, v)$ získáme z transformačního koeficientu $T(u, v)$ výrazem [16]

$$\hat{T}(u, v) = round\left[\frac{T(u, v)}{Q(u, v)}\right], \tag{2.6}$$

kde $Q(u,v)$ je prvek kvantizační matice

$$\mathbf{Q} = \begin{bmatrix} Q(0,0) & Q(0,1) & \cdots & Q(0,n-1) \\ Q(1,0) & Q(1,1) & \cdots & Q(,) \\ \vdots & \vdots & \ddots & \vdots \\ Q(n-1,0) & Q(n-1,1) & \cdots & Q(n-1,n-1) \end{bmatrix}. \tag{2.7}$$

Pokud $Q(u,v) > 2T(u,v)$, pak $\hat{T}(u,v) = 0$ a transformační koeficienty jsou kompletně useknuté nebo zahozené.

Zpětná kvantizace (dekvantizace) se pak řídí výrazem

$$\dot{T}(u,v) = \hat{T}(u,v) \cdot Z(u,v). \tag{2.8}$$

Ze vztahu je zřejmé, že dekvantizace již není ztrátová operace. Poznamenejme, že značení $\dot{T}$ namísto T ve výrazu (2.8) odkazuje právě na skutečnost, že rekonstruovaný signál není totožný s originálním; je pouze jeho aproximací.

Standardní kvantizační matice pro JPEG jsou dány komisí JPEG a byly odvozené empiricky. Pro kvalitu $Q = 50$ je kvantizační matice $\mathbf{Q}$ definována jako

$$\mathbf{Q}(50) = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \tag{2.9}$$

Každá jiná, škálovaná matice $\mathbf{Q}(Q)$ je odvozena jako

$$\mathbf{Q}(Q) = \text{round}(\mathbf{Q}(50) \cdot \alpha), \tag{2.10}$$

kde

$$\alpha = \begin{cases} 50/Q & \text{pro } Q \in \{1, \ldots, 49\} \\ 2 - 2Q/100 & \text{pro } Q \in \{51, \ldots, 99\} \end{cases} \tag{2.11}$$

Pokud je některý prvek $\mathbf{Q}(Q)$ nulový, uděláme z něj jedničku a znamená to, že daný koeficient nebude kvantován. Pro úplnost, matice $\mathbf{Q}(100)$ se skládá ze samých jedniček.[57]

2.1.4. Řazení koeficientů

Když kvantizací zajistíme, že se v určité oblasti budou nacházet pouze nulové koeficienty, můžeme toho následně využít během kódování. Například JPEG, kterým se budeme dále zabývat, používá tzv. Huffmanovo kódování s přesně danými, předpočítanými tabulkami. Jak systém funguje bude blíže vysvětleno v části 5.3. Pro teď je postačí říct, že využívá sekvencí nul následovaných nenulovým koeficientem. Takového uspořádání dosáhneme vhodným přeskládáním matice s kvantizačními koeficienty.

Jednou z možností je způsob tzv. *cik-cak* (zig-zag). Získáme tak 1D řadu koeficentů seřazených od nejnižších frekvencí (největších koeficientů) po nejvyšší frekvence (velmi malé až nulové koeficienty). Pořadí hodnot, které jsou řazené postupně od 0 do 63, udává tabulka 2.1.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

Tabulka 2.1: Tabulka cik-cak schématu.

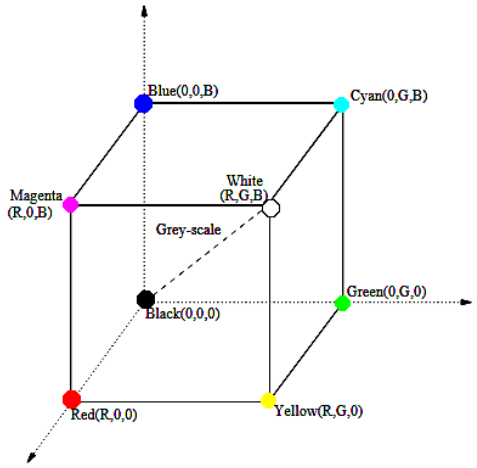
3. Barevné modely

I když budeme pracovat pouze s obrazy v odstínech šedi, jsou dnes barevné modely podstatnou součástí jakékoliv práce s digitálními obrazy. Modelů byla vytvořena celá řada (viz např. [22]) a většina z nich má nějaké své specifické využití. V souvislosti s JPEG jsou pro nás důležité zejména prostory barev označované jako RGB a YC_BC_R. RGB je nejběžnější model používaný pro zobrazování digitálního obsahu na displayích. Naproti tomu model YC_BC_R je zase velmi praktický třeba právě pro kódování a uchovávání obrazů na disku. Proto se na oba modely jejich vzájemný vztah podívejme podrobněji.

3.1. Model RGB

Model RGB se skládá ze tří základních barev, resp. barevných kanálů - červený, zelený a modrý (R, G, B; ang. red, green, blue). Všechny ostatní barvy vznikají skládáním jejich intenzit, které spadají do intervalu [0,255]. Hodnota 0 značí, že barevná složka není zastoupena vůbec. Např. [R,G,B] = [0,0,0] znamená, že není vyzařována žádná barva, na obrazovce je tedy černá. Naopak [R,G,B] = [255,255,255] se zobrazí jako bílá, protože všechny tři složky jsou zobrazeny v maximální míře.

Protože RGB nevyžaduje žádné další zpracování, stal se v počítačové technice dominantním barevným modelem.



Obrázek 3.1: Model RGB.

Jsou rozlišovány dva typy RGB modelu: *lineární barevný prostor RGB* a *nelineární barevný prostor RGB*. Přičemž lineární se obvykle značí jako „RGB“ a nelineární jako „R'G'B'“.[22]

3.1.1. Lineární barevný prostor RGB

Hodnoty lineárního RGB vznikají ve snímacích zařízeních (např. fotoaparáty) proporcionálně k vlnové délce zachytávaného světla. Proto jsou nezávislé na zařízení, čehož je někdy využíváno k přenosu konzistentních barev mezi různými zařízeními.

Tento prostor není vhodný pro numerickou analýzu a jen zřídka se používá pro reprezentaci obrazu. Například pro výsledky aplikace algoritmů JPEG nebo MPEG na lineární RGB ústí v nepoužitelné výsledky.[54]

3.1.2. Nelineární barevný prostor RGB

Nelineární hodnoty R'G'B' (v rozsahu $[0, 1]$) jsou získané transformací z lineárního RGB pomocí tzv. gamma korekce. Poté jsou obvykle konvertovány do běžně používaného rozsahu kanálů $[0, 255]$. Právě tato celočíselná data se pak ukládají v počítačích a jsou používána v aplikacích pro zpracování obrazu.

Transformační vztahy pro převod hodnot z lineárního prostoru do nelineárního a naopak (vždy v rozsahu $[0, 1]$) jsou formulovány např. v [54]. Vztahy pro převod lineárního RGB do nelineárního R'G'B' jsou:

$$R' = \begin{cases} 4,5\ R, & \text{když } R \leq 0,018 \\ 1,099 R^{\frac{1}{\gamma_C}} - 0,099 & \text{jinak} \end{cases} \tag{3.1}$$

$$G' = \begin{cases} 4,5\ G, & \text{když } G \leq 0,018 \\ 1,099 G^{\frac{1}{\gamma_C}} - 0,099 & \text{jinak} \end{cases} \tag{3.2}$$

$$B' = \begin{cases} 4,5\ B, & \text{když } B \leq 0,018 \\ 1,099 B^{\frac{1}{\gamma_C}} - 0,099 & \text{jinak} \end{cases} \tag{3.3}$$

γ_C je již zmíněný *gamma faktor* kamery nebo jiného vstupního zařízení. Hodnota, která je běžně užívána ve videokamerách je $\gamma_c = \frac{1}{0,45} (\simeq 2,22)$.

Vztahy pro převod nelineárních kanálů R'G'B' do lineárních RGB jsou:

$$R' = \begin{cases} \dfrac{R'}{4,5}, & \text{když } R' \leq 0,018 \\ \left(\dfrac{R'+0,099}{1,099}\right)^{\gamma_D} & \text{jinak} \end{cases} \tag{3.4}$$

$$G = \begin{cases} \dfrac{G'}{4,5}, & \text{když } R' \leq 0,018 \\ \left(\dfrac{G'+0,099}{1,099}\right)^{\gamma_D} & \text{jinak} \end{cases} \tag{3.5}$$

$$B = \begin{cases} \dfrac{B'}{4,5}, & \text{když } R' \leq 0,018 \\ \left(\dfrac{B'+0,099}{1,099}\right)^{\gamma_D} & \text{jinak} \end{cases} \tag{3.6}$$

a kde γ_D je v tomto případě gamma faktor zobrazovacího zařízení (obrazovky). Obvykle hodnota $\gamma_D \in [2,22; 2,45]$.

V literatuře, pokud se mluví o modelu „RGB“, myslí se většinou nelineární prostor.[54] V dalším textu se proto budeme držet tohoto značení.

3.1.3. Nevýhody RGB při zpracování obrazů

Model RGB je sice snadný na implementaci, ale neodpovídá způsobu, jakým barvy vnímá člověk. Barvy jsou navíc často závislé na konkrétním zařízení.[14] Ve vědeckém zkoumání není model RGB tak často používán (na rozdíl třeba od webových aplikací), protože odstín, jas a saturace jsou zobrazeny společně a nelze je snadno vyčlenit, takže některé detaily je v případě potřeby složité upravovat.[69] Navíc jsou R, G, B kanály přirozených obrazů silně korelované. Proto je často lepší pracovat s jasovou-chromatickou reprezentací, kde je informace dekorelovaná.[12]

3.2. Modely YUV a YCbCr

Barevný model YC_BC_R vychází z obecnějšího modelu YUV jako jeho škálovaná a ofsetová verze. Oba jsou používané pro televizní vysílání, ale zatímco YUV je analogový model, YC_BC_R se stal také standardem digitálním.[69]

3.2.1. Model YUV

Stejně jako zkratka RGB, i zkratky YUV či YC_BC_R modelu označují složky barevného spektra, použité v modelu. Y je jasová složka (luminance). Zobrazujeme jí jako spektrum v odstínech šedi. Složky U a V nesou barevnou a část jasové informace a jsou bipolární (mají jak záporné, tak kladné hodnoty).[14]

Stejně jako u normalizovaného RGB systému je černá barva reprezentována minimální hodnotou 0 a bílá maximální hodnotou 1. Pro zajímavost uveďme, že převod mezi RGB a YUV systémem je uskutečněn následujícími výrazy [22]:

$$
\begin{aligned}
Y &= 0,299 \cdot R' + 0,587 \cdot G' + 0,114 \cdot B' \\
U &= -0,147 \cdot R' - 0,289 \cdot G' + 0,436 \cdot B' = 0,492(B' - Y) \\
V &= 0,615 \cdot R' - 0,515 \cdot G' - 0,100 \cdot B' = 0,877(R' - Y) \\
R' &= Y + 1,140 \cdot V \\
G' &= Y - 0,395 \cdot U - 0,581 \cdot V \\
B' &= Y + 2,032 \cdot U
\end{aligned}
$$

3.2.2. Model YCbCr

YC_BC_R je prostor barev definovaný ve standardu ITU-R BT.601 [8], který byl vytvořený ITU (International Telecommunication Union), a je hojně používaný v evropských televizních signálech.[22] Kromě televizního vysílání našel své uplatnění také ve standardech jako je JPEG, MPEG, DVD nebo v zařízeních jako kamery, digitální televize a další.[69]

Y je stejně jako v modelu YUV jasová složka a je definována tak, že má 8-bitový rozsah $[16, 235]$. Složky C_B a C_R jsou chromatické a reprezentující rozdíl modré, resp. červené barvy s referenční hodnotou. Z toho plyne, že YC_BC_R není absolutním, ale relativním barevným prostorem. C_B a C_R jsou definovány s nominálním rozsahem $[16, 240]$. [22][33]

Převod z RGB prostoru se řídí následujícími vztahy. Analogové R'G'B' s normalizovaným rozsahem $[0, 1]$ se nejprve převádí na analogové YP_BP_R [33]:

$$
\begin{aligned}
Y' &= 0,299 \cdot R' + 0,587 \cdot G' + 0,114 \cdot B' \\
Pb &= -0,169 \cdot R' - 0,331 \cdot G' + 0,5 \cdot B' \\
Pr &= 0,5 \cdot R' - 0,419 \cdot G' - 0,081 \cdot B'
\end{aligned}
$$

R', G', B' jsou složky nelineárního modelu RGB. Značení Y' místo obyčejného Y odkazuje na tuto skutečnost. Převod do 8-bitového YC_BC_R je pak dán jako [33]

$$
\begin{aligned}
Y' &= 219 \cdot Y' + 16 \\
Cb &= 224 \cdot Pb + 128 \\
Cr &= 224 \cdot Pr + 128
\end{aligned}
$$

Předchozí vztahy však nejsou použitelné v JPEG, kde potřebujeme pracovat s plným 8-bitovým rozsahem o 256 ($[0, 255]$) úrovních pro každou komponentu. Podle standardu ITU-T T.871 [61] lze použít formule:

$$
\begin{aligned}
Y' &= 0,299 \cdot R' + 0,587 \cdot G' + 0,114 \cdot B' \\
Cb &= -0,169 \cdot R' - 0,331 \cdot G' + 0,5 \cdot B' + 128 \\
Cr &= 0,5 \cdot R' - 0,419 \cdot G' - 0,081 \cdot B' + 128
\end{aligned}
$$

A pro převod zpět z YC_BC_R do RGB:

$$
\begin{aligned}
R' &= Y' + 1,402(Cr - 128) \\
G' &= Y - 0,344(Cb - 128) - 0,714(Cr - 128) \\
B' &= Y + 1,772(Cb - 128)
\end{aligned}
$$

3.2.3. Typy podvzorkování

Důležitou vlastností modelu YC_BC_R je nezávislost jasové složka Y na odchylkách barevných složek signálu C_B a C_R. Další výhoda spočívá ve vlastnostech vnímání lidského oka, které je citlivé na změnu jasu, ale již méně na změnu barvy. Když ponecháme složku Y beze změny, pak při vhodné redukci (podvzorkováním) v chromatických složkách nedojde pro lidské oko k viditelným změnám v obrazu, ale zároveň se sníží nároky na paměť při ukládání či posílání obrazových signálů.[69]

Běžné vzorkovací poměry jsou 4:4:4, 4:2:2, 4:1:1 a 4:2:0. Nejpoužívanějšími jsou 4:2:2 a 4:2:0 a 4:2:0 speciálně pro JPEG.[55][56]

- **4:4:4** - Všechny pixely jsou vzorkovány, tj. jeden pixel má přiřazenou jednu barvu.
- **4:2:2** - Každým dvěma pixelům v horizontálním směru je přiřazena jedna barva. Celkový komprimační poměr je zde 33%.[56]
- **4:1:1** - v C_B a C_R je vzorkovaný každý čtvrtý pixel horizontálně.
- **4:2:0** - vzorkována je každá čtveřici pixelů 2×2, které je přiřazena její průměrná barva. Úspora je celkově 50%.[56]

Zpětné vzorkování na 4:4:4 se obvykle děje interpolací, kdy se dogenerují chybějící pixely C_B a C_R.[33]

(a) Orginální barevný obrázek. (b) Složka Y.

(c) Složka C_B. (d) Složka C_R.

Obrázek 3.2: Ukázka rozkladu RGB barevného obrázku do systému YC_BC_R. Jednotlivé složky jsou barevně zvýrazněné.

4. Diskrétní kosinová transformace

4.1. Diskrétní kosinová transformace

Diskrétní kosinová transformace (DCT, discrete cosine transform) se stala středobodem algoritmu JPEG. Jak bude ještě vysvětleno, je v podstatě základem pro jeho komprimační schopnosti. Proto se na ní teď zaměřme podrobněji.

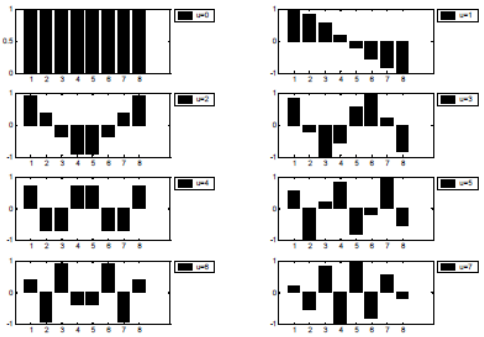
Koncept diskrétní kosinové transformace byl navržen Nasirem Ahmedem, T. Natarajanem a K. R. Raoem v roce 1972 a praktické výsledky byly prezentovány v roce 1974 v článku nazvaném „Discrete cosine transform".[5] Autoři v něm popsali formu DCT, která je dnes označována jako DCT typu II (DCT-II) a DCT typu III (DCT-III nebo také IDCT, viz následující kapitoly). Poznamenejme, že DCT byla již od počátku zamýšlena jako prostředek pro kompresi obrazu.[6][38]

Pro svou efektivitu a dobré výsledky se DCT v této oblasti uchytila a dnes je honě využívána jak pro kódování statických obrazů, tak i videí. Video lze totiž chápat jako sekvenci statických obrazů, které mezi sebou sdílejí velkou část informace (tj. vykazují vysokou korelaci). Mezi kompresní standardy, které využívají DCT patří například: JPEG, WebP, BPG, MPEG-1 - MPEG-4, WebM, JVT...[2][38]

4.1.1. Definice

Diskrétní kosinová transformace bezprostředně vychází z diskrétní Fourierovy transformace (DFT). Stejně jako DFT i DCT rozkládá vstupní signál na jednodušší signály postupně od nejnižších po nejvyšší frekvence. Ale zatímco DFT je založena na komplexních exponenciálách (viz např.[23]), DCT využívá systém kosinových funkcí. Pozitivním důsledkem je, že na rozdíl od DFT nedochází k nespojitostem na hranicích signálu.

Jednotlivé kosinové vlnové křivky (waveforms) nazýváme *kosinové bázové funkce* a tvoří ortogonální systém.[40] Jsou tedy nezávislé; žádná z bázových funkcí nemůže být vyjádřena jako kombinace ostatních bázových funkcí. To znamená, že násobením jakékoliv vlnové křivky s jakoukoliv další a následnou sumací přes všechny vzorkovací body dostaneme nulovou skalární hodnotu; násobením dané vlny se sebou samou a následnou sumací zase dostaneme konstantní skalární hodnotu.[38]



Obrázek 4.1: Bázové funkce DCT. Převzato z [39].

Co se týče samotného odvození DCT z DFT, můžeme jádro (transformation kernel) kosinové transformace v podstatě chápat jako reálnou část jádra Fourierovy transformace.[7]

[40][43] Dle [40] lze totiž komplexní jádro DFT v jednorozměrném případě rozložit na sudou reálnou a lichou imaginární složku

$$\varphi_{ux} = \cos\left(2\pi\frac{xu}{N}\right) + j\sin\left(2\pi\frac{xu}{N}\right). \tag{4.1}$$

Ani jedna z těchto částí ale nepřestavuje v oboru reálných funkcí úplný (ortogonální) systém bázových funkcí. Pokud však bude obrazová funkce sudá, resp. lichá, potom bude Fourierova transformace podle věty o symetrii (viz [40]) obsahovat pouze kosinové, resp. sinové, složky a pro takové funkce bude pro kosinová, resp. sinová část tvořit úplný systém.

Je vhodné poznamenat, že úplným systémem zde myslíme takový systém ortogonálních funkcí, kde na daném intervalu neexistuje jiná funkce, která by vyhovoval podmínce ortogonality.

Sudou funkci, označme ji $g(x)$, získáme vytvořením sudého prodloužení (rozšíření) funkce $f(x)$ kolem pevně daného středu x. Takových vhodných prodloužení pro odvození jádra DCT je více. Nejpoužívanější z nich je $2N$-bodové.

Nechť $x(n), 0 \leq x \leq N$ je N-bodová vstupní posloupnost, která představuje diskrétní signál. Pak její $2N$-bodové sudé prodloužení $y(n)$ se středem v bodě $n = -1/2$ definujeme vztahy

$$y(n) = \begin{cases} x(n) & \text{pro } 0 \leq n < N-1, \\ x(2N-x-1) & \text{pro } N \leq n < 2N-1. \end{cases} \tag{4.2}$$

Jak už bylo řečeno, protože $y(n)$ je sudé, jeho Fourierova transformace bude obsahovat pouze kosinové členy. Výpočtem této DFT a dalšími úpravami (viz [40]) jsme pak schopní vyjádřit jádro diskrétní kosinové transformace ve tvaru

$$\varphi_u = \sum_{n=0}^{N-1} \cos\frac{\pi(2n-1)u}{2N} \tag{4.3}$$

pro $0 \leq n \leq N-1$.[40]

Na základě tohoto jádra uveďme nyní definice a základní formy DCT v jedné dimenzi. Z praktického hlediska to znamená transformaci aplikovatelnou například na signály zvuku.

Mějme posloupnost $x(n)$, reprezentující hodnotu vstupního diskrétního signálu o délce N v bodech $n = 0, 1, \ldots, N$. Dále definujme pro $u = 0, \ldots, N$ posloupnost transformačních koeficientů $C(u)$, která odpovídá vždy dané formě kosinové transformace (DCT-I - DCT-IV). Pak dle [43] definujeme jednotlivé verze přímé (dopředné) DCT a odpovídající inverzní (zpětné) transformace (IDCT, inverse discrete cosine transform) takto:

DCT-I:

$$C^I(u) = \sqrt{\frac{2}{N}}\,\alpha_u \sum_{n=0}^{N} \alpha_n x(n) \cos\left(\frac{\pi n u}{N}\right) \quad \text{pro } u = 0, 1, \ldots, N \tag{4.4}$$

$$x(n) = \sqrt{\frac{2}{N}}\,\alpha_n \sum_{u=0}^{N} \alpha_u C^I(u) \cos\left(\frac{\pi n u}{N}\right) \quad \text{pro } u = 0, 1, \ldots, N \tag{4.5}$$

DCT-II:

$$C^{II}(u) = \sqrt{\frac{2}{N}}\,\alpha_u \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi(2n+1)u}{2N}\right) \quad \text{pro } u = 0, 1, \ldots, N-1 \tag{4.6}$$

$$x(n) = \sqrt{\frac{2}{N}} \sum_{u=0}^{N-1} \alpha_u C^{II}(u) \cos\left(\frac{\pi(2n+1)u}{2N}\right) \quad \text{pro } n = 0, 1, \ldots, N-1 \tag{4.7}$$

DCT-III:

$$C^{III}(u) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} \alpha_n x(n) \cos\left(\frac{\pi n(2u+1)}{2N}\right) \quad \text{pro } u = 0, 1, \ldots, N-1 \tag{4.8}$$

$$x(n) = \sqrt{\frac{2}{N}}\, \alpha_n \sum_{u=0}^{N-1} C^{III}(u) \cos\left(\frac{\pi n(2u+1)}{2N}\right) \quad \text{pro } n = 0, 1, \ldots, N-1 \tag{4.9}$$

DCT-IV:

$$C^{IV}(u) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi(2n+1)(2u+1)}{4N}\right) \quad \text{pro } u = 0, 1, \ldots, N-1 \tag{4.10}$$

$$x(n) = \sqrt{\frac{2}{N}} \sum_{u=0}^{N-1} C^{IV}(u) \cos\left(\frac{\pi(2n+1)(2u+1)}{4N}\right) \quad \text{pro } n = 0, 1, \ldots, N-1 \tag{4.11}$$

Pro koeficienty α_k platí

$$\alpha_k = \begin{cases} \dfrac{1}{\sqrt{2}} & \text{když } k = 0 \text{ nebo } k = N, \\ 1 & \text{když } k \neq 0 \text{ a } k \neq N. \end{cases} \tag{4.12}$$

Nejpoužívanější z těchto variant, zejména ve 2D prostoru (viz dále), je DCT-II. Ta je zároveň v mnoha zdrojích (např.[38][40][44]) udávána jako jediná. Využívá ji také JPEG. Proto se, pokud nebude řečeno jinak, další text vztahuje právě k této formulaci.

Drobnou úpravou je možné získat ještě další formulaci DCT. Normalizační faktor $\sqrt{2/N}$, který se objevuje v přímé i inverzní transformaci, lze sloučit jako $2/N$ a přesunout pouze do přímé nebo inverzní transformace. Tak lze například DCT-II definovat [40] také jako

$$C^{II}(u) = \frac{2}{N}\, \alpha_u \sum_{n=0}^{N-1} x(n) \cos\left(\frac{\pi(2n+1)u}{2N}\right) \quad \text{pro } u = 0, 1, \ldots, N-1. \tag{4.13}$$

A k ní inverzní transformace bude mít tvar

$$x(n) = \sum_{u=0}^{N-1} \alpha_u C^{II}(u) \cos\left(\frac{\pi(2n+1)u}{2N}\right) \quad \text{pro } n = 0, 1, \ldots, N-1. \tag{4.14}$$

Tedy výrazy pro $C^{II}(u)$ v (4.6) a (4.13) spolu souvisejí pomocí měřítka $\sqrt{2/N}$.

Pokud má vstupní signál více než N vzorkovacích bodů, pro které chceme DCT vypočítat, pak může být rozdělen do menších částí o délce N a DCT aplikujeme na každou

část zvlášť. Pro každý takový výpočet se hodnoty bodů bázových funkcí nezmění, mění se pouze hodnoty posloupnosti $x(n)$ v rámci každé části. Tato vlastnost nám umožňuje předpočítat bázové funkce a poté pouze násobit s jednotlivými částmi. To významným způsobem redukuje potřebné matematické operace (tj. násobení a sčítání) pro výpočet DCT a tedy celkově zvyšuje výpočetní efektivitu.

Transformační koeficienty, které získáme aplikací DCT, mají své specifické označení. Zvolme nyní $u = 0$ a dosaďme do rovnice (4.6). Dostáváme $C(u = 0) = \sqrt{\frac{1}{N}} \sum_{n=0}^{N-1} x(n)$. První transformační koeficient je tedy průměrnou hodnotou vstupního signálu. Tato hodnota se nazývá *DC koeficient*. Všechny ostatní transformační koeficienty jsou nazývány *AC koeficienty* a obsahují bázové kosinové funkce s postupně zvyšujícími se frekvencemi. Toto označení pochází z historického využití DCT pro analýzu elektrických obvodů se stejnosměrnými a střídavými proudy (angl. *direct* a *alternating currents*).[38]

4.2. 2D DCT

Při zpracování obrazů si s transformací definovanou pro 1D prostor samozřejmě nevystačíme. I když, jak bude ukázáno později, v případě DCT to může být při správně zvoleném způsobu aplikace dostačující.

DCT ve dvou-dimenzionálním prostoru je přímým rozšířením případu v 1D. Místo signálu o délce N daného posloupností $x(n)$, použijeme nyní matici $\mathbf{X}$ o velikosti $M \times N$ vzorků. Tedy $X(m, n)$ udává například hodnotu pixelu na pozici $[m, n]$. Diskrétní kosinovou transformaci $\mathbf{C}$ matice $\mathbf{X}$ (ve formulaci odpovídající (4.6)) definujeme jako [43]

$$C(u,v) = \frac{2}{\sqrt{MN}} \alpha_u \alpha_v \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} X(m,n) \cos\left[\frac{\pi(2m+1)u}{2N}\right] \cos\left[\frac{\pi(2n+1)v}{2N}\right]. \quad (4.15)$$

pro $u = 0, 1, \ldots, M-1$, $v = 0, 1, \ldots, N-1$. Inverzní transformace je pak definována výrazem

$$X(m,n) = \frac{2}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} \alpha_u \alpha_v C(u,v) \cos\left[\frac{\pi(2m+1)u}{2N}\right] \cos\left[\frac{\pi(2n+1)v}{2N}\right], \quad (4.16)$$

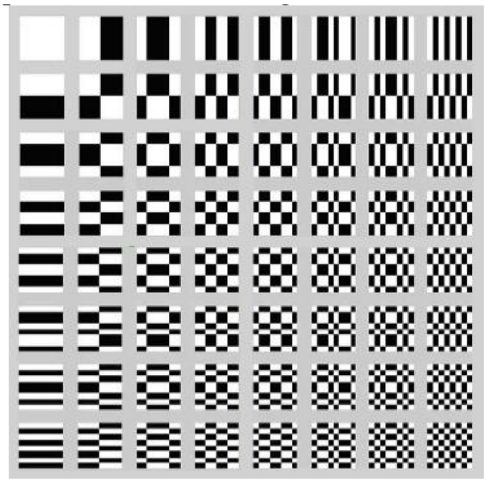
kde $m = 0, 1, \ldots, M-1$, $n = 0, 1, \ldots, N-1$. Pro koeficienty α_k platí

$$\alpha_k = \begin{cases} \frac{1}{\sqrt{2}} & \text{když } k = 0 \\ 1 & jinak. \end{cases} \quad (4.17)$$

Bázové funkce v dvou-dimenzionálním prostoru mohou být generovány násobením horizontálně orientovaných 1D bázových funkcí s těmi samými orientovaným vertikálně. Stejně jako v jedné dimenzi bázové funkce vykazují postupné zvyšování frekvence jak v horizontálním, tak ve vertikálním směru.[40]

Toho se využívá zejména v aplikacích pro kompresi obrazu. U typického obrazu je většina vizuálně významných informací o obraze soustředěna pouze v menším množství transformačních koeficientech zastupující nízké frekvence.[44] Informace obsažená ve vysokých frekvencích často není pro lidské oko ani pozorovatelná. Jejím odstraněním vhodnou

metodou (tj. nějakou formou kvantizacace) nepoškodíme vizuální kvalitu obrazu a zároveň redukujeme množství koeficientů, se kterými kompresní algoritmus musí dále pracovat. To vede jak k nižší výpočetní náročnosti, tak k větší efektivitě komprimace.



Obrázek 4.2: Bázové funkce DCT. Převzato z [18].

Na obrázku 4.2 znázorňuje čtverec v horním levém rohu bázovou funkci vzniklou násobením DC složky v grafu 4.1 s její transpozicí. Tím dostáváme konstantní hodnotu a nazýváme ji stejně jako v jedno-dimenzionálním případě DC koeficient. Stejně tak zbývající koeficienty označujeme jako AC koeficienty.

4.3. Vlastnosti DCT

Uveďme nyní některé praktické vlastnosti kosinové transformace, popsané v [38], a jejich význam při využití transformace v aplikacích zpracování obrazů.

4.3.1. Dekorelace

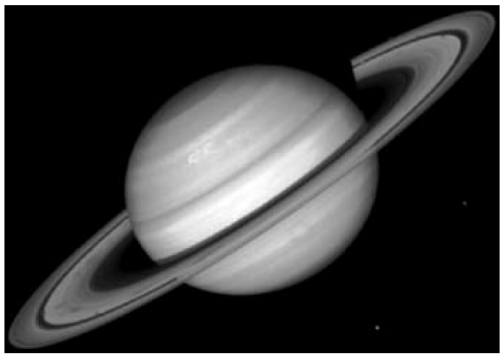
Důležitou výhodou obrazové transformace je odstranění redundance (tj. nadbytečnosti informace), kterou mezi sebou sdílejí sousední pixely. To vede k nekorelovaným transformačním koeficientům, které mohou být kódovány nezávisle.

Zjevně, amplituda autokorelace po aplikaci DCT je velmi malá. Odtud můžeme vyvozovat, že DCT vykazuje výborné dekorelační vlastnosti.

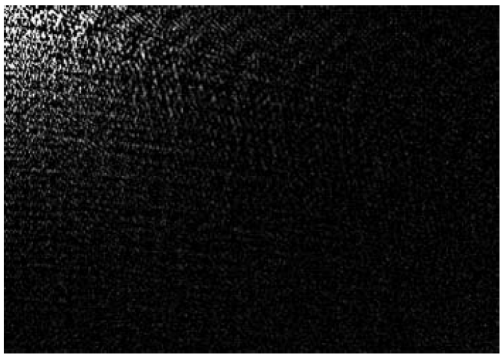
4.3.2. Zhuštění energie

Účinnost transformace může být přímo hodnocena na základě její schopnosti „zabalit“ vstupní data do co nejmenšího počtu transformačních koeficientů. To znamená, že většina informace o obrazu se bude nacházet pouze v koeficientech odpovídajících nízkým frekvencím (viz sekce 4.2).

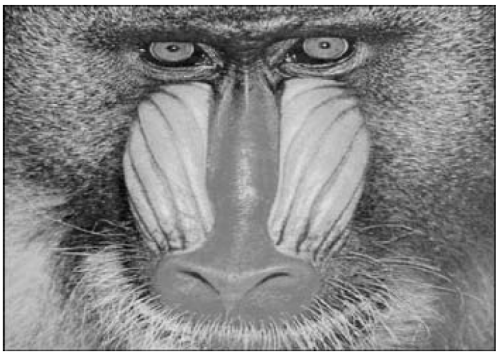
DCT vykazuje vynikající schopnost zhuštění energie pro vysoce korelované obrazy. Navíc se ukazuje, že dosahuje optima, když se korelace obrazu blíží k 1. To znamená, že DCT poskytuje téměř optimální dekorelaci takových obrazů.




(a) Obrázek Saturn a jeho frekvenční spektrum.

(b) Obrázek Strom a jeho frekvenční spektrum.




(c) Obrázek Opice a jeho frekvenční spektrum.

Obrázek 4.3: Porovnání obrazů před a po JPEG kompresi.

Protože u nekorelovaných obrazů jsou rozdíly mezi jasovými složkami větší než u korelovaných, mají nekorelované obrazy více obsahu ve vysokých frekvencích a tedy i větší rozptyl energie (tj. transformačních koeficientů). U korelovaných obrazů, tak jak můžeme pozorovat na obrázcích 4.3 je energie shromážděna do oblasti s nízkými frekvencemi u levého horního rohu.

Z obrázků 4.3 je patrné, že obrazy s velkými oblastmi s pomalu měnícími se hodnotami jasu obsahují pouze málo detailů a jejich transformační koeficienty spadají převážně do oblasti s nízkými frekvencemi. Označíme je tedy jako nízkofrekvenční obrazy s malým množstvím prostorových detailů. DCT poskytuje velmi dobré zhuštění energie v těchto nízkofrekvenčních oblastech. Obrazy s množstvím hran (tj. ostrými změnami úrovní jasu) mohou být označeny jako vysokofrekvenční obrazy s nízkým prostorovým obsahem. Na-

konec, obrazy s vysokou úrovní šumu mají jak významný podíl jak vysokofrekvenčního, tak i prostorového obsahu.

DCT obrazů, které vykazují periodicitu (např. obraz sinové vlny), obsahuje impulzy s amplitudami odpovídajícími váze dané frekvence v původním obrazu vlnové křivky.

4.3.3. Separabilita

Separabilita DCT ve dvou-dimenzionálním prostoru má svůj význam především při vytváření počítačových algoritmů pro samotný výpočet transformace. Původní rovnici (4.15) lze vyjádřit jako

$$C(u,v) = \alpha_u \alpha_v \sum_{m=0}^{M-1} \cos\left[\frac{\pi(2m+1)u}{2N}\right] \sum_{n=0}^{N-1} X(m,n) \cos\left[\frac{\pi(2n+1)v}{2N}\right]. \tag{4.18}$$

Tuto vlastnost nazýváme *separabilita* a platí i pro inverzní transformaci.

Hlavní výhodou je, že nyní lze $C(u,v)$ počítat ve dvou následných krocích jako 1D DCT aplikované na řádky a sloupce tvořené pixely obrazu.

4.3.4. Ortogonalita

Transformační jádro DCT lze stejně jako u jiných transformací vyjádřit v maticové podobě. Transformační matice $\mathbf{T}$ o velikosti $M \times M$ je dána [36]

$$\mathbf{T}_{mn} = \begin{cases} \sqrt{\dfrac{1}{M}} & \text{pro } m = 0, \quad 0 \leq n \leq M-1, \\ \sqrt{\dfrac{2}{M}} \cos\left[\dfrac{\pi m(2n+1)}{2N}\right] & \text{pro } 1 \leq m \leq M-1, \quad 0 \leq n \leq M-1. \end{cases} \tag{4.19}$$

Odtud 2D DCT a 2D IDCT mohou být napsány v maticové formě jako

$$\mathbf{C} = \mathbf{T}\mathbf{x}\mathbf{T}^{-1}, \tag{4.20}$$

$$\mathbf{x} = \mathbf{T}^{-1}\mathbf{C}\mathbf{T}. \tag{4.21}$$

Protože DCT patří mezi sinusoidální unitární transformací [34], je její transformační matice ortonormální. Pro nesingulární reálné a ortonormální matice platí, že jejich inverze je rovná jejich transpozici. Tedy $T^{-1} = T^T$.[43]

Toto je další velmi užitečná vlastnost DCT. Umožňuje nám totiž transformační matici předpočítat nezávisle na vstupních datech. Tím může být v některých případech dosaženo podstatně větší výpočetní efektivity.

4.4. Rychlý výpočet DCT

Od rozšíření DCT do oblasti zpracování obrazů bylo vytvořeno mnoho metod, které se snaží o rychlejší a efektivnější výpočet DCT. Uveďme jako příklad způsob, který využívá algoritmus rychlé Fourierovy transformace (FFT, ang. Fast Fourier Transfrom).

Algoritmus staví na faktu, že posloupnost $x(n)$ v (4.6) může být vyjádřena jako suma sudé a liché posloupnosti

$$\tilde{x}(n) = x_S(n) + x_L(n),$$

$x_S(n) = x(2n) = \tilde{x}(n)$, $x_L(n) = x(2n+1) = \tilde{x}(N-1-n)$, kde $0 \leq n \leq \left(\frac{N}{2}\right) - 1$.

Sumace ve výrazu (4.6) může být rozdělena, abychom získali

$$\begin{aligned} C(u) &= \alpha_u \left\{ \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n) \cos\left[\frac{\pi(4n+1)u}{2N}\right] + \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x(2n+1) \cos\left[\frac{\pi(4n+3)u}{2N}\right] \right\} \\ &= \alpha_u \left\{ \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x_S(n) \cos\left[\frac{\pi(4n+1)u}{2N}\right] + \sum_{n=0}^{\left(\frac{N}{2}\right)-1} x_L(n) \cos\left[\frac{\pi(4n+3)u}{2N}\right] \right\} \\ &= \alpha_u \sum_{n=0}^{N-1} \tilde{x}(n) \cos\left[\frac{\pi(4n+3)u}{2N}\right] \\ &= \mathrm{Re}\left[\alpha_u \exp^{-j\pi u/2N} \sum_{n=0}^{N-1} \tilde{x}(n) \exp^{-j2\pi un/N} \right] \end{aligned} .$$

Sumace v posledním kroku je diskrétní Fourierova transformace. Pro inverzní transformaci můžeme spočítat $x(2n) = \tilde{x}(2n)$ a sudé body lze vypočítat z výrazu $x(2n+1) = \tilde{x}[2(N-1-n)]$ pro $0 \leq x \leq \left(\frac{N}{2}\right) - 1$.

4.5. Redukce entropie

Dekorelační schopnost DCT by měla také snížit míru entropie obrazu. Entropii můžeme popsat jako míru stupně nahodilosti informace obrazu nebo jinak řečeno jako míru neuspořádanosti. Důležitým důsledkem redukce entropie je redukce počtu bitů potřebných k uložení (resp. zakódování) obrazu v paměti. Entropii obrazu $H(x)$ definujeme jako

$$H(X) = -\sum_{i=0}^{N-1} p_i \log p_i, \tag{4.22}$$

kde N je počet úrovní šedé (256 pro 8-bitové obrazy) a p_i je hodnota pravděpodobnostní funkce, že pixel bude mít odstín šedé i.[52] Pravděpodobností funkce zde referuje na normalizovaný histogramu obrazu.

V tabulce 4.1 jsou uvedené entropie obrazů z odstavce 4.3.2. „DCT(M%)“ znamená, že po provedení DCT bylo zachováno pouze prvních M procent celkového počtu transformačních koeficientů. K výpočtu entropie zde bylo využito funkce MATLAB `entropy`.

obrázek obrázek	entropie obrázku	entropie DCT(75%)	entropie DCT(50%)	entropie DCT(25%)
Saturn	4,11	1,91	0,75	0,22
Strom	5,70	1,14	0,62	0,21
Opice	7,35	0,99	0,59	0,21

Tabulka 4.1: Tabulka entropií obrazů z oddílu 4.3.2 a entropií jejich DCT po redukci koeficientů.[38]

Tabulka ukazuje, že aplikace DCT snižuje entropii všech obrazů. Redukce entropie je tím výraznější, čím menší procento koeficientů zachováme. Tento výsledek naznačuje, že zahození některých detailů v obrazu, které jsou reprezentovány vysokými frekvencemi, se může významně podílet na snížení celkové entropie. A tedy na snížení průměrného počtu bitů nutných k zakódování obrazu.

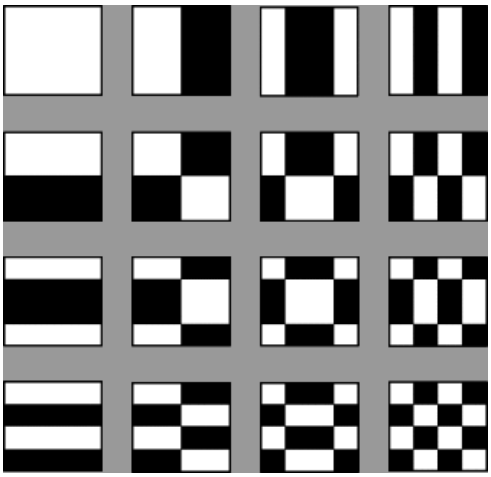
Tato redukce koeficientů při ztrátovém kompresi je implementována v podobě kvantizéru. Viz část 2.1.3.

4.5.1. Stručné porovnání DCT a dalších transformací

KLT je obecně optimálním řešením ve smyslu komprese informace. To proto, že minimalizuje střední kvadratickou chybu z rovnice (11.1) pro jakýkoliv vstupní obraz a pro jakýkoliv počet zachovaných koeficientů. Bázové funkce KLT jsou totiž odvozované ze statistických vlastností obrazových dat a mohou se adaptovat. Ačkoliv, KLT transformační jádro není obecně separabilní a tedy musí být prováděno násobení plnou maticí. Jinými slovy, nelze využít rychlou (ve smyslu FFT) předpočítanou transformaci a odvození odpovídajících bází pro každý pod-blok obrázku vyžaduje, vzhledem k záměru využití transformace, nerozumné výpočetní zdroje. I když byly pro rychlý výpočet KLT navrhnuty některé algoritmy, je stále celková náročnost tvyšší než u DCT či DFT algoritmů. Z tohoto důvodu je většina obrazových komprimačních metod založena DCT, která poskytuje dobrý kompromis právě mezi schopností komprimovat informaci a výpočetní náročností.

DFT oproti WHT sice vykazuje lepší vlastnosti pro kompresi obrazů, ale stále není zcela ideální transformací. Transformační jádro DFT je lineární, separabilní a symetrické. Proto, stejně jako u DCT, má fixní bázové funkce a implementace rychlých algoritmů je možná. Vykazuje také dobrou dekorelaci a kompresi energie. Na druhou stranu je DFT komplexní transformací, takže vynucuje kódování informace jak o velikosti, tak i o fázi.

Bylo prokázáno, že DCT obecně poskytuje lepší kompresi energie než DFT (a WHT) pro většinu přirozených obrazů. Proti použití DFT svědčí také fakt, že její implicitní periodicita dává vzniknout nespojitostem na hranách, které se projeví značným nárůstem obsahu ve vyšších frekvencích. Způsobuje také tzv. Gibbsův jev, díky kterému body na ostrých hranách nabývají chybných hodnot. Zviditelní se tak hranice pod-obrazů a v rekonstruovaném obraze se to projeví jako blokové artefakty. DCT tento efekt redukuje, protože její implicitní $2n$-bodová periodicita sama o sobě nevytváří takové hranové nespojitosti.[16][38]



Obrázek 4.4: Hranice DFT a DCT a vznik nespojitostí. Převzato z [16].

4.6. Alplikace DCT v prostředí MATLAB

MATLAB umožňuje počítat 2D DCT funkcemi `dct2` a `dctmtx`. První z nich je určený pro velká vstupní data a využívá algoritmus založený na FFT. Druhá funkce využívá transformační matici DCT a je vhodná pro menší vstupy o velikostech jako 8×8 nebo 16×16.[44]

5. JPEG

I s rozvojem novějších algoritmů ztrátové komprese (jako například PGF (2000)[1], WebP (2010) [2] či dalších variant jako JPEG 2000 (2000), JPEG XL (2021) [37] apd.) je původní JPEG stále honě využívaným algoritmem a formátem pro kompresi a kódování obrázků. Ačkoli jde o ztrátový formát, kdy se kódováním ztrácí část kvality původního obrázku, jsou soubory JPEG obvykle dostatečně kvalitní pro širokou škálu digitálních aplikací. Stal se natolik oblíbeným, že je dnes zaručena jeho kompatibilita prakticky se všemi obrazovými procesory a prohlížeči.[3]

Používá se zejména tam, kde je cílem zachovat pro lidské oko vysokou vizuální kvalitu, ale zároveň ukládat soubory s „rozumnou“ velikostí.

Označení JPEG není původně pouze označení kodeku (z ang. codec, tj. zkratka označení systému „coder - decoder“) nebo přípona souborů s obrázky, ale je zkratkou pro pracovní skupinu Joint Photographic Experts Group, sdružující odborníky z Mezinárodní organizace pro normalizaci (ISO) a Mezinárodní elektrotechnické komise (IEC). Slovo ”Joint”v názvu skupiny však neodkazuje na spojení pracovníků ISO a IEC, ale na skutečnost, že činnosti JPEG jsou výsledkem další spolupráce s Mezinárodní telekomunikační unií (ITU), která mimo jiné spravuje technické specifikace telekomunikačních systémů.

Skupina JPEG stále funguje a se schází obvykle čtyřikrát ročně, aby projednávala a vytvářela normy pro kódování digitálních reprezentací obrazů. [37]

Poznamenejme, že v souborových systémech počítačů a mobilních zařízení se můžeme setkat se dvěma téměř totožnými příponami souborů kódovaných algoritmem JPEG a to „.jpeg“ a „.jpg“. Nejde o odlišné formáty. Jediným důvodem, proč má druhá varianta pouze tři znaky, je, že rané verze systému Windows vyžadovaly třípísmennou příponu pro názvy souborů.[3] Struktura i vlastnosti obou typů zůstávají funkčně totožné.

5.1. Historie

Uveďme nyní ve stručnosti historii a vývojové verze systému JPEG. Jejich podrobné specifikace lze nalézt například na oficiálních stránkách skupiny JPEG, viz [37].

5.1.1. JPEG 1

Jako JPEG 1 se nazývá původní verze normy (ISO/IEC 10918) [24] z roku 1992 (s poslední verzí z roku 1994), jejíž vývoj začal již v roce 1986. Pro zajímavost uveďme, že i přes rozšíření a obecnou oblibu tohoto formátu ISO/EIC nikdy neposkytla referenční software, který by demonstroval správnou implementaci normy. [37]

Z této základní formy vychází i tato práce a celým algoritmem se budeme zabývat v následujících kapitolách.

5.1.2. JPEG 2000

JPEG 2000 (ISO/IEC 15444) byl oficiálně jako standard přijat v roce 2000 [25] a zveřejněn v roce 2001. Záměrem bylo vytvořit vylepšenou verzi původního formátu JPEG s lepším kompresním poměrem a vizuálně kvalitnějšími obrazy.

Výhodami jsou například: ztrátová i bezztrátová komprese, vyšší rychlost než u původního formátu, kvalitnější výstupy, bitová hloubku až 38 bitů/vzorek, škálovatelnost rozlišení a kvality, implementace na CPU, GPU a FPGA a další.

Bohužel se tato nová verze JPEG nesetkala s úspěchem. I když je standard stále udržován, obdržel jistá vylepšení (nový algoritmus blokového kódování (ISO/IEC 15444-15) [29] nebo nová sada prostorových waveletových transformací (ISO/IEC 15444-17), které umožňují kódování médií s tvrdými diskontinuitami) a je podporován některými aplikacemi, je stále jeden z nejméně rozšířených.

Příčinou je především rozdílný kód JPEG 2000, který není zpětně kompatibilní s předchozí verzí. To znamená, že starší verze hardwaru nebyly schopné s touto verzí pracovat. V článku [4] se lze dočíst, že z tohoto důvodu nebyl formát dobře přijat ani vývojáři, kteří se zdráhali implementovat jak starý, tak i nový kód. To samozřejmě ovlivnilo povědomí uživatelů a zároveň s tím například i ochotu výrobců fotoaparátů JPEG 2000 požívat, dokud se nestane populárnějším. Důležitou roli hrály i větší nároky na paměť RAM, které v té době nebyly zanedbatelné. [37]

5.1.3. JPEG XR

V roce 2009 byl představen JPEG XR (ISO/IEC 29199) [26]. Tvůrci jej popisují jako standard poskytující praktickou technologii kódování pro širokou škálu aplikací s vynikajícími kompresními schopnostmi, který zároveň zachovává požadavek jednoduchosti implementace kodérů a dekodérů.

Primárním zamýšleným použitím JPEG XR je reprezentace souvislých tónových statických obrazů, jakými jsou fotografické snímky. Zvláštním zaměřením je podpora obrazových aplikací s vysokým dynamickým rozsahem (HDR, ang. high dynamic range), které se před vznikem tohoto standardu začali významně rozšiřovat.

JPEG XR poskytuje rozsáhlou sadu dalších funkcí, jako například: nízké nároky na výpočetní a paměťové zdroje (vyžaduje pouze malý počet celočíselných operací zpracování pro kódování i dekódování), struktura zpracování signálu vhodná pro paralelní zpracování, bezeztrátovou a ztrátovou kompresi (přičemž v obou případech používá stejné operace zpracování signálu, což je důležité zejména pro vestavěné aplikace), podpora alfa kanálu, bitově přesné výsledky dekodéru pro obrazové formáty s pevnou i plovoucí řádovou čárkou (ačkoli se pro interní zpracování používá pouze celočíselná aritmetika) a další…

Standard podporuje širokou škálu formátů kódování barev včetně monochromatického, RGB, CMYK a n-složkového kódování s různou bitovou hloubkou: Při ztrátové kompresi jsou podporovány 32bitové formáty. Pro ztrátovou i bezeztrátovou kompresi jsou podporovány 8bitové a 16bitové formáty a další specializované zabalené bitové formáty.

Zpracování je založeno na makroblocích 16×16 v transformační doméně, které mohou, ale nemusí ovlivňovat odpovídající překrývající se oblasti v prostorové doméně (přičemž vlastnost překrývání kontrolována kodérem). [37]

5.1.4. JPEG XT

JPEG XT (ISO/IEC 18477), standardizovaný v roce 2015 [27], specifikuje řadu zpětně zcela kompatibilních rozšíření staršího standardu JPEG (ISO/IEC 10918-1). Zahrnuje řešení komprese obrazů s vyšší bitovou hloubkou (9 až 16 bitů), zobrazování s vysokým

dynamickým rozsahem, kódování s plovoucí desetinnou čárkou, bezeztrátové komprese, kódování alfa kanálu a rozšiřitelný formát souborů založený na JFIF. [37][59]

5.1.5. JPEG XS

Norma JPEG XS (ISO/IEC 21122) z roku 2019 [28] specifikuje kompresní technologii s end-to-end latencí (tj. čas potřebný k přenesení paketu od zdroje k cíli) s pouze několika řádky při nízké implementační složitosti. Tím jsou umožněny například hardwarové implementace, které nevyžadují externí paměť. Kodek navíc umožňuje velmi přesné řízení přenosové rychlosti, aby přesně odpovídal dostupné šířce pásma (např. gigabitového Ethernetu) a nabízí zpoždění mezi koncovými body rovnající se zlomku snímku. Konstrukce JPEG XS nabízí různé stupně paralelismu umožňující efektivní implementaci na různých platformách jako jsou FPGA, ASIC, CPU a GPU.

Tato verze vyniká také vysokou vícegenerační robustností a je optimalizována zejména pro vizuální bezztrátovou kompresi pro přirozené i syntetické obrazy. Typické kompresní poměry jsou až 10:1, ale mohou být i vyšší v závislosti na povaze obrazu nebo požadavcích cílové aplikace. Standard rovněž nabízí bezeztrátovou kompresi obrazových dat s přesností až 12 bitů na složku. Konečně, jsou podporovány různé formáty pixelů včetně surového Bayerova formátu, klasického RGB a typické reprezentace YUV (viz 3.2.1), všechny až do 16 bitů na složku.

Všechny tyto vlastnosti umožňují použití JPEG XS v aplikacích, které dříve přenášely nekomprimovaná obrazová data (např. pro automobilový průmysl nebo špičkovou profesionální fotografii). Různé systémy tak mohou nabízet vyšší rozlišení a snímkovací frekvenci při zachování vizuálně bezeztrátové kvality. [37]

5.1.6. JPEG XL

JPEG XL (ISO/IEC 18181) je relativně novou verzí systému JPEG. Formát souboru byl totiž standardizován v roce 2021 a kódovací systém v roce 2022. [30][31]

JPEG XL je optimalizován zejména pro responzivní webové prostředí tak, aby se obsah dobře vykresloval na široké škále zařízení, a pro práci s profesionálními fotografiemi. Podporuje rozsáhlý barevný gamut (tj. množina zobrazitelných barev z daného prostoru barev), vysoký dynamický rozsah a vysokou bitovou hloubkou. Dále obsahuje funkce jako animace, alfa kanály, vrstvy, náhledy, bezeztrátové, progresivní kódování a další. Dochází tak k výraznému rozšíření možností použití kodeku.
Snadný přechod ze starší, či do starší verze formátu JPEG je zajištěn několika funkcemi.

Stávající soubory JPEG lze bezeztrátově překódovat do formátu JPEG XL, čímž se výrazně sníží jejich velikost. Tím se snižují také náklady na ukládání, protože servery mohou ukládat jediný soubor JPEG XL, který slouží klientům JPEG i JPEG XL. Soubory lze obnovit do přesně stejného souboru JPEG, čímž je zase zajištěna zpětná kompatibilita se stávajícími aplikacemi.

JPEG XL nabízí výrazně lepší kvalitu obrazu, kompresní poměry i kratší specifikaci než starší verze JPEG. Zachovává vysokou věrnost zdrojovému obrazu, přičemž kompresní poměr se obvykle pohybuje v rozmezí 20:1 až 50:1. Je navržen pro výpočetně efektivní kódování a dekódování pomocí softwarových implementací bez nutnosti další hardwarové akcelerace, a to i v mobilních zařízeních.[37]

5.2. Sekvenční a progresivní standard

Ještě než popíšeme samotný algoritmus JPEG je dobré zmínit a vysvětlit rozdíl mezi jeho dvěma běžně používanými kódovacími standardy. Jsou jimi *progresivní* (progressive) JPEG a *sekvenční* (resp. základní; ang. baseline) JPEG. Oba jsou založené na klasickém algoritmu, ale každý zpracovává obraz trochu jiným způsobem.

Progresivní standard představuje „základní“ algoritmus tak, jak bude popsán v této kapitole. V tomto případě je výstupní soubor uložen jako jeden „sken“ vstupního obrazu shora dolů a je tak také dekódován.

Vývoj progresivního standardu přímo souvisí s rozvojem internetu a webových stránek. Umožňuje totiž najednou načíst celý obraz nejprve v horší kvalitě a až poté ho postupně vykresluje do plného rozlišení. To znamená, že uživatel má možnost vidět přibližnou podobu obrazu v podstatě okamžitě, bez čekání na jeho úplné načtení.

Progresivní JPEG rozděluje soubor do několika skupin - „skenů“ - podle frekvencí koeficientů DCT (viz 4). Skeny jsou načítány od nízkých frekvencí k vysokým, takže detaily v obrázku přibývají postupně. Velikost souboru na disku je podobná tomu zpracovanému sekvenčním standardem. Nevýhodou ovšem je, že každý sken je přibližně stejně výpočetně náročný, jako celý sekvenční soubor JPEG.

Kromě schopnosti progresivního zobrazení jsou progresivní JPEG a sekvenční JPEG v podstatě identické a fungují dobře na stejných druzích obrázků.

Existuje také kódovací standard nazývaný *hierarchický*. Ten se ale kromě velmi specifických aplikaci v praxi téměř nepoužívá a proto se jím nebudeme více zabývat.[15][42][68]

5.3. Algoritmus

Algoritmus JPEG je vhodný pro většinu kompresních aplikací, které pracují s plnobarevnými obrazy (fotografií). Poskytuje reverzibilní bezztrátový kódovací systém (i když, jak už bylo řečeno, komprese je sama o sobě ztrátová) s jednoduchou implementací a nízkou energetickou náročností.

Vstupní a výstupní datová velikost je omezena na 8, někdy 12 bitů na kanál a pixel. Kvantizované hodnoty DCT jsou omezeny na 11 bitů. Bez přílišné vizuální ztráty kvality umožňuje poměr komprese barevného obrazu až 25:1.[16][56]

Základem kompresních vlastností systému JPEG je DCT, následná kvantizace a konečně tvorba binárního kódu s variabilní délkou založená na Huffmanově kódování. Nejde ale o jediné kroky, které se podílejí na kódování a kompresi. Před tím než můžeme vůbec aplikovat DCT musíme zajistit, aby měl vstupní obraz vhodnou velikost, byl převeden do vyhovujícího barevného spektra a rozčleněn na menší bloky.

Podívejme se na všechny zmíněné kroky nyní podrobněji a s aplikací na konkrétním příkladu.

5.3.1. Transformace barev

Obrazy standardně vstupují do algoritmu v barevném prostoru RGB. Ten ale není z důvodů uvedených v kapitole 3 příliš vhodný. Převodem do prostoru YC_BC_R získáme hodnoty pixelů v rozmezí $[0, 255]$ pro tři jeho složky - jasovou Y, a dvě chromatické C_B a C_R. Protože, jak už bylo řečeno, je lidské oko méně citlivé na změnu barvy než na

změnu jasu, můžeme chromatické složky, opět s odkazem na kapitolu 3, podvzorkovat a zbavit se tak bez újmy na kvalitě části nedůležité informace. Nejčastěji používané typy podvzorkování jsou 4:2:0 a 4:2:2.[56] Tím v podstatě dochází k první úspoře na objemu dat, která musíme zakódovat a uložit. Jasovou složku ponecháme beze změny.

5.3.2. Posun hodnot pixelů

Hodnoty pixelů ve všech třech složkách se obvykle posouvají o hodnotu 2^{k-1}, kde 2^k je maximální množství úrovní jasu v obrazu. Pokud uvažujeme 8-bitový obraz o 256 úrovních jasu, pak tyto složky posouváme o $2^7 = -128$ (tj. posunuté hodnoty spadají do intervalu [-128, 127]). Posunem dochází k centrování hodnot okolo nuly. Tento krok je pro blokovou matici 8×8 to samé jako odečítat 1024 od DC koeficientu po provedení DCT.

Hodnoty AC koeficientů se tímto posunem nijak nezmění. Zmenší se však DC koeficient, což znamená, že bude potřeba méně bitů k jeho zakódování. V dnešní době velkých pamětí je však význam tohoto kroku diskutabilní. Vzhledem k tomu, že DC koeficienty se kódují jako diference (viz dále) a pouze koeficient prvního bloku se kóduje jako jeho skutečná hodnota, není úspora bitů vzhledem k celku významná.

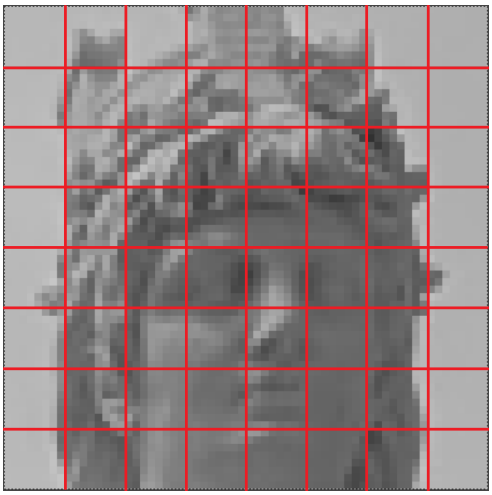
5.3.3. Rozdělení do bloků

Dále je obraz rozdělený do bloků o velikosti 8×8 pixelů, které jsou zpracovávány jednotlivě zleva doprava, ze shora dolů. Pokud rozměry obrazu nejsou násobkem 8, tak se obraz vhodným způsobem rozšíří přidáním potřebných pixelů na pravé a spodní hraně. Tj. ve směru kladných souřadnic x a y. Možností rozšíření je několik [13][56]:

1. Rozšíření o nulové hodnoty. Tím ale můžeme vytvořit také více vysokofrekvenčního obsahu díky velkému skoku v hodnotách původních a nových pixelů. Mohou tak vznikat silné artefakty na hranicích.

2. Opakování koncové hodnoty.

3. Zrcadlení odpovídajícího počtu pixelů na okrajích.



Obrázek 5.1: Výřez z úvodního obrázku o rozměrech 64×64 pixelů se znázorněným dělením na bloky 8×8 pixelů.

Mějme nyní nějaký náhodný blok pixelů, jehož hodnoty jasu tvoří matici

$$\mathbf{I} = \begin{bmatrix} 154 & 213 & 217 & 198 & 199 & 179 & 97 & 192 \\ 138 & 149 & 194 & 172 & 162 & 157 & 98 & 113 \\ 204 & 124 & 117 & 130 & 115 & 125 & 95 & 58 \\ 198 & 122 & 99 & 56 & 80 & 114 & 82 & 48 \\ 106 & 74 & 123 & 68 & 71 & 103 & 70 & 45 \\ 52 & 53 & 134 & 96 & 69 & 98 & 59 & 46 \\ 111 & 69 & 134 & 120 & 73 & 95 & 50 & 48 \\ 192 & 193 & 176 & 143 & 86 & 95 & 46 & 47 \end{bmatrix}.$$

Nyní hodnoty posuneme o -128. Tedy

$$\mathbf{I}_S = \mathbf{I} - 128 = \begin{bmatrix} 26 & 85 & 89 & 70 & 71 & 51 & -31 & 64 \\ 10 & 21 & 66 & 44 & 34 & 29 & -30 & -15 \\ 76 & -4 & -11 & 2 & -13 & -3 & -33 & -70 \\ 70 & -6 & -29 & -72 & -48 & -14 & -46 & -80 \\ -22 & -54 & -5 & -60 & -57 & -25 & -58 & -83 \\ -76 & -75 & 6 & -32 & -59 & -30 & -69 & -82 \\ -17 & -59 & 6 & -8 & -55 & -33 & -78 & -80 \\ 64 & 65 & 48 & 15 & -42 & -33 & -82 & -81 \end{bmatrix}.$$

5.3.4. DCT a kvantizace

Následně se pro každý blok spočítá 2D diskrétní kosinová transformace (viz odstavec 4.2). Matice transformačních koeficientů je pak kvantizována (tj. vydělena kvantizační maticí) a koeficienty jsou zaokrouhleny na celá čísla (viz vztah (2.6)). Kvantizací se zachovají pouze důležité koeficienty o nízkých frekvencích, ostatní, které mají jen malý vliv na kvalitu obrazu, jsou vynulovány.

Standardní kvantizační matici JPEG $\mathbf{Q}$ (viz 2.1.3) lze škálovat volbou kvality komprese $Q \in \{1, \ldots, 100\}$. Ovlivňujeme tak množství nulovaných koeficientů. Zřejmě, čím více koeficientů odstraníme, tím lepší bude komprese, ale horší vizuální kvalita obrazu. Vzhledem k systému $\mathrm{YC_BC_R}$ jsou součástí komprimovaného souboru dvě kvantizační matice. Jedna pro jasovou složku a druhá pro obě barevné složky.[56]

Pro naši vstupní matici $\mathbf{I}_S$ bude matice transformačních koeficientů

$$\mathbf{C}_S = DCT(\mathbf{I}_S) = \begin{bmatrix} -106,0 & 182,3 & -52,3 & 16,1 & -20,5 & 12,1 & 86,6 & -8,7 \\ 192,9 & -38,9 & -6,1 & 6,9 & 25,2 & -45,8 & -8,0 & -0,3 \\ 178,9 & 37,4 & -45,7 & -77,6 & 19,7 & -47,6 & -4,8 & -31,0 \\ -23,7 & -96,3 & -58,5 & -68,9 & -14,2 & -27,2 & 25,0 & 5,8 \\ 53,5 & 44,5 & 50,5 & 7,5 & -9,5 & -39,0 & -7,4 & -3,3 \\ -4,6 & -2,5 & 30,1 & 25,1 & 21,7 & 1,9 & 0,6 & -5,1 \\ 15,2 & 13,9 & 5,2 & -0,3 & 3,2 & -3,5 & -12,8 & -9,2 \\ -2,2 & -3,0 & -3,0 & 4,1 & 15,3 & 13,0 & 0,1 & -2,0 \end{bmatrix}.$$

Pro porovnání ukažme i DCT vstupní matice **I** bez posunutých hodnot

$$\mathbf{C} = DCT(\mathbf{I}) = \begin{bmatrix} 918,0 & 182,3 & -52,3 & 16,1 & -20,5 & 12,1 & 86,6 & -8,7 \\ 192,9 & -38,9 & -6,1 & 6,9 & 25,2 & -45,8 & -8,0 & -0,3 \\ 178,9 & 37,4 & -45,7 & -77,6 & 19,7 & -47,6 & -4,8 & -31,0 \\ -23,7 & -96,3 & -58,5 & -68,9 & -14,2 & -27,2 & 25,0 & 5,8 \\ 53,5 & 44,5 & 50,5 & 7,5 & -9,5 & -39,0 & -7,4 & -3,3 \\ -4,6 & -2,5 & 30,1 & 25,1 & 21,7 & 1,9 & 0,6 & -5,1 \\ 15,2 & 13,9 & 5,2 & -0,3 & 3,2 & -3,5 & -12,8 & -9,2 \\ -2,2 & -3,0 & -3,0 & 4,1 & 15,3 & 13,0 & 0,1 & -2,0 \end{bmatrix}.$$

Zřejmě, pro DC koeficienty platí $DC(\mathbf{C}) - DC(\mathbf{C}_S) = 918,0 - (-106,0) = 1024$. Zbytek transformačních koeficientů má totožné hodnoty, jak bylo řečeno v odstavci 5.3.2.

Kvantizací matice $\mathbf{C}_S$ dostáváme matici kvantizačních koeficientů

$$\mathbf{T} = \begin{bmatrix} -7 & 17 & -5 & 1 & -1 & 0 & 2 & 0 \\ 16 & -3 & 0 & 0 & 1 & -1 & 0 & 0 \\ 13 & 3 & -3 & -3 & 0 & -1 & 0 & -1 \\ -2 & -6 & -3 & -2 & 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix},$$

kde např. DC koeficient je vypočítaný jako

$$T(0,0) = \text{round}\left(\frac{\mathbf{C}_S(0,0)}{\mathbf{Q}(0,0)}\right) = \text{round}\left(\frac{-106}{16}\right) = -7.$$

5.3.5. Kódování entropie

Po kvantizaci je matice přeskládána za použití cik-cak vzoru (viz oddíl 2.1.4). Koeficienty jsou seřazené kvalitativně podle zvyšující se prostorové frekvence. Tím na konci sekvence dostáváme řadu nulových hodnot. Toho procedura JPEGu využívá použitím Huffmanova kódování. Stejně jako v případě kvantizační matice, i zde poskytuje komise JPEG předdefinované kódovací tabulky.[16]

V našem případě dostáváme po přeskládání matice **T** 1D pole

[-7 17 16 13 -3 -5 1 0 3 -2 3 -6 -3 0 -1 0 1 -3 -3 2 0 0 0 1 -2 0 -1 2 0 0 -1 ...
... 0 0 1 0 0 0 0 0 0 0 0 0 -1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0].

Kódování DC koeficientů

DC koeficienty jsou kódovány relativně vždy vzhledem k DC koeficientu předcházejícího bloku: $DIFF = DC_i - DC_{i-1}$, kde, jak už bylo řečeno, pro první koeficient DC_1 je zakódována jeho reálná hodnota.

V úvodu jsme uvedli, že dynamický rozsah koeficientů DCT pro 8-bitové pixely je 11 bitů. Po diferenci ale může vzrůst až na 12 bitů, pokud bychom přecházeli přes celý rozsah. Proto se požívá kódovací tabulka s 12 kategoriemi (viz tabulka 5.2). Pro reálné obrazy, kde se barvy obvykle mění pomalu, jsou však mnohem pravděpodobnější malé hodnoty.

diference DC koef./ hodnota AC koef.	kategorie DC	kategorie AC
0	0	N/A
-1, 1	1	1
-3, -2, 2, 3	2	2
-7 ,..., -4, 4,..., 7	3	3
-15,..., -8, 8,..., 15	4	4
-31,..., -16, 16,..., 31	5	5
-63,..., -32, 32,..., 63	6	6
-127,..., -64, 64,..., 127	7	7
-255,..., -128, 128,..., 255	8	8
-511,..., -256, 256,..., 511	9	9
-1023,..., -512, 512,..., 1023	A	A
-2047,..., -1024, 1024,..., 2047	B	B
-4095,..., -2048, 2048,..., 4095	C	C
-8191,..., -4096, 4096,..., 8191	D	D
-16383,..., -8192, 8192,..., 16383	E	E
-32767,..., -16384, 16384,...,32767	F	N/A

Tabulka 5.1: Kategorie pro DC a AC koeficienty podle velikosti koeficientu.[9][16]

kategorie DC	kódové slovo	délka kódu
0	00	2
1	010	3
2	011	3
3	100	3
4	101	3
5	110	3
6	1110	4
7	11110	5
8	111110	6
9	1111110	7
10	11111110	8
11	111111110	9

Tabulka 5.2: Kódovací slova pro DC kategorie.[9][16]

Zakódována je kategorie podle svého kódu v Huffmanově tabulce (tabulka 5.2) a konkrétní číslo. Pokud je koeficient záporný, použije se 1' doplněk. Např. koeficient s hodnotou -5 zakódujeme jako $(101)' = 010$.[16][56]

Postup při kódování DC koeficientu v našem případě vypadá takto. Předpokládejme, že transformovaný a kvantizovaný DC koeficient předchozího bloku byl 6. Pak diference bude $-7 - 6 = -13$. V Huffmanových tabulkách tato hodnota spadá do kategorie DC 4 (tab. 5.1) a odpovídá kódovému slovu 101. Číslo 13 v binárním zápisu je 1101. Protože je ale koeficient záporný, musíme použít doplněk 0010. Výsledné kódové slovo pro DC koeficient je pak 1010010.

Kódování AC koeficientů Nenulové AC koeficienty jsou kódovány za použití „variabilně" dlouhého kódu, který je definován kombinací *run-size* kategorie koeficientu a jeho

vlastní hodnotou. *Run* je počet nul předcházející nenulové hodnotě, kterou kódujeme, a *size* je AC kategorige, do které koeficient spadá (viz 5.1). Kódovací slovo pro každou kombinaci run-size (R|S) lze opět vyčíst z Huffmanových tabulek, viz 5.3 a 5.4. Pokud je hodnota koeficientu záporná, opět vezmeme 1' doplněk.

Speciálními symboly v tabulkách 5.3 a 5.4 jsou EOB a ZRL. EOB (z ang. end-of-block) značí, že zbytek bloku obsahuje již pouze nulové hodnoty. V Huffmanových tabulkách má tento symbol své specifické kódovací slovo 1010 pro R|S = 0|0. ZRL (zero-run-lenght) označující 15 nul v řadě následovaných další nulou. Tedy například run 20 nul následovaný 5 bychom zakódovali jako (ZRL)(4|3)101. ZRL najdeme v tabulce AC koeficientů jako R|S = 15|0 a binárním vyjádřením 11111111001.[9][56]

run/size	kódové slovo	délka kódu	run/size	kódové slovo	délka kódu
0/0	**1010 (EOB)**	4			
0/1	00	3	8/1	111111000	9
0/2	01	4	8/2	111111111000000	17
0/3	100	6	8/3	1111111110110110	19
0/4	1011	8	8/4	1111111110110111	20
0/5	11010	10	8/5	1111111110111000	21
0/6	1111000	12	8/6	1111111110111001	22
0/7	11111000	14	8/7	1111111110111010	23
0/8	1111110110	18	8/8	1111111110111011	24
0/9	1111111110000010	25	8/9	1111111110111100	25
0/A	1111111110000011	26	8/A	1111111110111101	26
1/1	1100	5	9/1	111111001	10
1/2	11011	8	9/2	1111111110111110	18
1/3	1111001	10	9/3	1111111110111111	19
1/4	111110110	13	9/4	1111111111000000	20
1/5	11111110110	16	9/5	1111111111000001	21
1/6	1111111110000100	22	9/6	1111111111000010	22
1/7	1111111110000101	23	9/7	1111111111000011	23
1/8	1111111110000110	24	9/8	1111111111000100	24
1/9	1111111110000111	25	9/9	1111111111000101	25
1/A	1111111110001000	26	9/A	1111111111000110	26
2/1	11100	6	A/1	111111010	10
2/2	11111001	10	A/2	1111111111000111	18
2/3	1111110111	13	A/3	1111111111001000	19
2/4	111111110100	20	A/4	1111111111001001	20
2/5	1111111110001001	21	A/5	1111111111001010	21
2/6	1111111110001010	22	A/6	1111111111001011	22
2/7	1111111110001011	23	A/7	1111111111001100	23
2/8	1111111110001100	24	A/8	1111111111001101	24
2/9	1111111110001101	25	A/9	1111111111001110	25
2/A	1111111110001110	26	A/A	1111111111001111	26
3/1	111010	7	B/1	1111111001	10
3/2	111110111	11	B/2	1111111111010000	18
3/3	111111110101	14	B/3	1111111111010001	19
3/4	1111111110001111	20	B/4	1111111111010010	20
3/5	1111111110010000	21	B/5	1111111111010011	21
3/6	1111111110010001	22	B/6	1111111111010100	22
3/7	1111111110010010	23	B/7	1111111111010101	23
3/8	1111111110010011	24	B/8	1111111111010110	24
3/9	1111111110010100	25	B/9	1111111111010111	25
3/A	1111111110010101	26	B/A	1111111111011000	26

Tabulka 5.3: Kompletní tabulka kódovacích slov pro DC kategorie - část 1.[16][9]

run/size	kódové slovo	délka kódu	run/size	kódové slovo	délka kódu
4/1	111011	7	C/1	1111111010	11
4/2	1111111000	12	C/2	1111111111011001	18
4/3	1111111110010110	19	C/3	1111111111011010	19
4/4	1111111110010111	20	C/4	1111111111011011	20
4/5	1111111110011000	21	C/5	1111111111011100	21
4/6	1111111110011001	22	C/6	1111111111011101	22
4/7	1111111110011010	23	C/7	1111111111011110	23
4/8	1111111110011011	24	C/8	1111111111011111	24
4/9	1111111110011100	25	C/9	1111111111100000	25
4/A	1111111110011101	26	C/A	1111111111100001	26
5/1	1111010	8	D/1	11111111000	12
5/2	11111110111	12	D/2	1111111111100010	18
5/3	1111111110011110	19	D/3	1111111111100011	19
5/4	1111111110011111	20	D/4	1111111111100100	20
5/5	1111111110100000	21	D/5	1111111111100101	21
5/6	1111111110100001	22	D/6	1111111111100110	22
5/7	1111111110100010	23	D/7	1111111111100111	23
5/8	1111111110100011	24	D/8	1111111111101000	24
5/9	1111111110100100	25	D/9	1111111111101001	25
5/A	1111111110100101	26	D/A	1111111111101010	26
6/1	1111011	8	E/1	1111111111101011	13
6/2	111111110110	13	E/2	1111111111101100	18
6/3	1111111110100110	19	E/3	1111111111101101	19
6/4	1111111110100111	20	E/4	1111111111101110	20
6/5	1111111110101000	21	E/5	1111111111101111	21
6/6	1111111110101001	22	E/6	1111111111110000	22
6/7	1111111110101010	23	E/7	1111111111110001	23
6/8	1111111110101011	24	E/8	1111111111110010	24
6/9	1111111110101100	25	E/9	1111111111110011	25
6/A	1111111110101101	26	E/A	1111111111110100	26
7/1	11111010	9	**F/0**	**11111111001 (ZRL)**	12
7/2	111111110111	13	F/1	1111111111110101	17
7/3	1111111110101110	19	F/2	1111111111110110	18
7/4	1111111110101111	20	F/3	1111111111110111	19
7/5	1111111110110000	21	F/4	1111111111111000	20
7/6	1111111110110001	22	F/5	1111111111111001	21
7/7	1111111110110010	23	F/6	1111111111111010	22
7/8	1111111110110011	24	F/7	1111111111111011	23
7/9	1111111110110100	25	F/8	1111111111111100	24
7/A	1111111110110101	26	F/9	1111111111111101	25
			F/A	1111111111111110	26

Tabulka 5.4: Kompletní tabulka kódovacích slov pro DC kategorie - část 2.[16][9]

AC koeficienty z našeho příkladu jsou:

[…17 16 13 -3 -5 1 0 3 -2 3 -6 -3 0 -1 0 1 -3 -3 2 0 0 0 1 …
…-2 0 -1 2 0 0 -1 0 0 1 0 0 0 0 0 0 0 0 0 -1 EOB].

Způsob jejich kódování ukazuje tabulka 5.5.

AC koeficient	odpovídající R|S	bitový zápis koeficientu
17	0|5	10001
16	0|5	10000
13	0|4	1101
-3	0|2	00 ← 11
…	…	…
1	2|1	1
-1	9|1	0 ← 1
EOB	0|0	1010

Tabulka 5.5: Tabulka cik-cak schématu.

A výsledný bitový proud i s DC složkou pak pro celý blok bude:

1010010//**1101**10001/**1101**10000/**1011**1101/**01**00/…/**11100**1/**1111110010**/**1010**

Symbol „/“ byl přidán pouze pro přehlednost, přičemž „//“ odděluje DC koeficient. Tučně zvýrazněná část udává kódovací slovo z Huffmanových tabulek.

Velikost celého zakódovaného bloku je $7 + 134 = 141$ bitů (včetně vynechané části AC koeficientů). Původní matice měla 64 hodnot, každou po 8 bitech. To znamená 512 bitů. Kompresní poměr pro tento blok je tedy asi 3,63:1 nebo jinak vyjádřeno 2,2bpp.

Kromě uvedených Huffmanových tabulek pro DC a AC koeficienty existují ještě další dvě pro chromatické složky. Uživatel si může také vytvořit svoje vlastní, které mohou být lépe přizpůsobeny požadavkům konkrétní aplikace a obrazům.[16]

5.3.6. Dekódování

Pro jasovou i obě barevné složky je postup dekódování přesně opačný ke kódování. Dekodér nejprve znovu vytvořit kvantizační koeficienty, ze kterých jsme vytvořili komprimovaný bitový proudu. Protože binární sekvence kódovaná za pomocí Huffmanových tabulek je okamžitě a jediněčně dekódovatelná, lze tento krok jednoduše provést pouze hledáním odpovídajících hodnot v tabulkách.[16] Bloky pak vynásobíme kvantizační maticí, na koeficienty aplikujme inverzní DCT, posuneme hodnoty zpátky o 2^k a nadvzorkujeme barevné složky. Nakonec převedeme obraz z YC_BC_R do zvoleného barevného modelu.

5.3.7. Algoritmem vytvořené artefakty

Výše uvedeným postupem získáme „původní“ hodnoty pixelů. Obraz však stejný není a často jsou na něm patrné „vady“, vzory, které na původním obrazu nebyli. Je to dáno jednak zaokrouhlováním během kódování, ale zejména kvantizací, kde jsme odstranili

velkou část vysokofrekvenčních složek signálu. Tyto vzniklé chyby odpovídají blokům 8×8, se kterými jsme pracovali.

Specifické chyby vznikají také podvzorkováním barevných složek C_B a C_R. Zde také zpracováváme bloky 8×8 pixelů, ale v původním obrázku odpovídají blokům o velikosti 16×16 pixelů.

Jak už bylo řečeno na začátku, cílem této práce bylo pokusit se tyto chyby odstranit aplikací hranového detektoru a proximálních algoritmů. V následujících kapitolách se budeme této problematice podrobněji věnovat.

6. Detekce hran

6.1. Cannyho algoritmus detekce hran

Detekce hran je považována za velmi důležitou součást výzkumu v oblasti zpracování obrazů a bylo na toto téma napsáno množství studií a prací. Hranu můžeme popsat jako křivku, která sleduje cestu rychlé změny intenzity jasu v obrazu. Obecně lze tvrdit, že hrany v obraze existují zejména mezi dvěma objekty, objektem a pozadím a dvěma odlišnými oblastmi. Účelem je tedy výrazně snížit množství dat v obrazu a zároveň zachovat strukturální vlastnosti, které lze využít pro další práci s obrazem. [11][45][62]

Hlavní částí algoritmů pro detekci hran jsou různé diferenciální operátory prvního řádu jako Robertův gradientní operátor, Prewittové operátor nebo Sobelův operátor a operátory druhého řádu jako Laplaceův a LOG operátor. [62]

V této práci je využit algoritmus původně vyvinutý Johnem F. Cannym v roce 1986. Přestože je poměrně starý, stal se pro svou relativní jednoduchost a efektivnost jedním ze standardních metod detekce hran a jeho různé formy se stále používají ve výzkumu. [11][62]

Cannyho metoda se od ostatních metod detekce hran liší zejména tím, že používá dva různé prahy pro detekci silných a slabých hran (viz dále). Slabé hrany zahrnuje do výstupu pouze v případě, že jsou přímo spojeny se silnými. U této metody je proto méně pravděpodobné, že bude ovlivněna šumem a spolehlivěji odhalí skutečné slabé hrany. [45] I přesto, jak bude vysvětleno později, má původní Cannyho algoritmus stále některé nedokonalosti. Byly však vytvořené vylepšené verze (např. [62][63][67]), které tyto nedostatky významně redukují.

Cannyho algortimus je založený na optimalizačním algoritmu a na třech striktních kritériích detekce hran. Tato kritéria jsou optimální zejména pro tzv. „stupňovité hrany“. [10][11][62]

1. **Poměr signálu k šumu:** Pravděpodobnost detekce skutečných hranových bodů by měla být maximalizována, zatímco pravděpodobnost falešné detekce nehranových bodů by měla být minimalizována. Tomu odpovídá maximalizace poměru signálu k šumu (SNR, z ang. signal-to-noise ratio). Hodnota se počítá takto:

$$SNR = \frac{|\int_{-W}^{+W} G(-x)h(x)dx|}{\sigma\sqrt{\int_{-W}^{+W} h^2(x)dx}}, \tag{6.1}$$

   kde $G(x)$ je hranová funkce a $h(x)$ je impulzní odezva filtru o šířce W. σ zastupuje střední kvadratickou chybu Gaussova šumu.

2. **Přesnost lokalizace:** Detekované hrany by měly být co nejblíže skutečným hranám. Přesnost určení polohy hrany je definována jako

$$L = \frac{|\int_{-W}^{+W} G'(-x)h'(x)dx|}{\sigma\sqrt{\int_{-W}^{+W} h'^2(x)dx}}, \tag{6.2}$$

   kde $G'(-x)$ a $h'(x)$ jsou derivacemi funkcí $G(x)$ a $h(x)$. Čím větší je přesnost L, tím lepší dává algoritmus výsledky.

3. **Jedinečnost odpovídající hrany:** Jedna skutečná hrana by neměla vést k více než jedné detekované hraně. Aby byla jedinečnost zajištěna, měla by průměrná vzdálenost $D(f')$ průsečíku derivace impulzní odezvy s nulou splňovat následující výraz:

$$D(f') = \pi \left(\frac{\int_{-\infty}^{+\infty} h'^2(x)dx}{\int_{-W}^{+W} h''^2(x)dx} \right)^{\frac{1}{2}}, \tag{6.3}$$

kde $h''(x)$ je druhá derivace funkce $h(x)$.

Tradiční Cannyho algoritmus detekce hran je rozdělen do pěti samostatných kroků [11][67]:

1. **Vyhlazení:** Rozmazání obrazu za účelem odstranění šumu. Používá se Gaussův filtr.

2. **Nalezení gradientů:** Výpočet velikosti a směru gradientů v obrazu. Hrany by měly být označeny tam, kde mají gradienty velkou velikost.

3. **Potlačení nemaximálních hodnot:** Jako hrany by měla být označena pouze lokální maxima. Tím dochází ke zpřesnění hran.

4. **Dvojité prahování:** Potenciální hrany a jejich síla jsou vybrány pomocí dolního a horního prahu.

5. **Sledování hran pomocí hystereze:** Konečné hrany jsou určeny potlačením všech hran, které nejsou spojeny s velmi jistou (silnou) hranou.

Každý krok je popsán v následujících podkapitolách.

6.1.1. Testovací obrázek

Vstupní obraz byl předzpracován. Předzpracování zahrnuje: - Určení oblasti zájmu (ROI, ang. region of interest), která obsahuje pouze bílé pozadí vedle pumpy, a ořez obrazu na tuto oblast. - Převod do stupňů šedi, aby se omezily výpočetní nároky. - Roztažení histogramu tak, aby obraz využíval celou škálu šedi. Tento krok nemusí být nutný, ale je zařazen jako kompenzace automatické úpravy světla v použité web-kameře. [11]

6.2. Vyhlazování

Všechny přirozené obrazy (tj. snímky pořízené pomocí kamery) budou nevyhnutelně obsahovat určité množství šumu. Jeho redukcí zabráníme tomu, aby byl mylně považován za hrany. Proto se obraz nejprve vyhladí použitím Gaussova filtru [11]

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{x^2 + y^2}{2\sigma^2} \right), \tag{6.4}$$

kde σ je směrodatná odchylka sloužící k řízení stupně vyhlazení obrazu.[10]

Účinek vyhlazení testovacího obrazu tímto filtrem je na obrázku [obr].

6.3. Nalezení gradientů

Cannyho algoritmus detekce hran v podstatě vyhledává místa, na kterých je změna jasu (resp. hodnoty odstínu šedé) obrazu lokálně nejvýznamnější. V jedné dimenzi to znamená najít lokální maxima pomocí první derivace. Dvojrozměrnou variantou je gradient, který určuje jak míru změny, tak její směr. Ve vyhlazeném obraze se gradienty určují aplikací parciálních derivací prvního řádu ve směrech x a y na sousedící oblast 2×2 (resp. 3×3) pixelů. Mějme funkci složek

$$\mathbf{G} = \nabla f(x, y) = [G_x, G_y] = \left[\frac{\partial f(x, y)}{\partial x}, \frac{\partial f(x, y)}{\partial y} \right]. \tag{6.5}$$

[35]

Velikost gradientu, tedy maximální rychlost nárůstu funkce $f(x, y)$ na jednotku vzdálenosti ve směru $\mathbf{G}$, je dána vztahem

$$G = \sqrt{G_x^2 + G_y^2}. \tag{6.6}$$

Azimut gradientu je definován jako

$$\theta(x, y) = \arctan \left[\frac{G_y}{G_x} \right], \tag{6.7}$$

kde úhel θ se měří vzhledem k ose x. [35]

6.3.1. Numerická aproximace

U digitálních obrazů se derivace v rovnici (6.5) aproximují pomocí diferencí. Nejjednodušší aproximace gradientu je

$$G_x \cong f(i, j + 1) - f(i, j) G_y \cong f(i, j) - f(i + 1, j. \tag{6.8}$$

Tu lze realizovat pomocí jednoduchých konvolučních masek:

$$G_x = [-1, 1], G_y = \begin{matrix} 1 \\ -1 \end{matrix} \tag{6.9}$$

V tomto případě je aproximace počítaná v bodě $[i, j + \frac{1}{2}]$ pro G_x a v bodě $[i + \frac{1}{2}, j]$ pro G_y. Korektní určení hran však vyžaduje, aby s gradienty počítaly ve stejném bodě.[35] Proto Cannyho algoritmus využívá oblast o velikosti 2×2 pixelů. Pokud tuto oblast vyjádříme v maticovém zápisu [67]

$$I = \begin{bmatrix} f(x, y) & f(x, y + 1) \\ f(x + 1, y) & f(x + 1, y + 1) \end{bmatrix},$$

pak aproximaci gradientů v obou směrech zapíšeme jako

$$G_x = [f(x, y + 1) - f(x, y)]/2 + [f(x + 1, y + 1) - f(x + 1, y)]/2, \tag{6.10}$$
$$G_y = [f(x, y) - f(x + 1, y)]/2 + [f(x, y + 1) - f(x + 1, y + 1)]/2. \tag{6.11}$$

a jejich odpovídající konvoluční masky jsou:

$$G_x = \begin{matrix} -1 & 1 \\ -1 & -1 \end{matrix}, \tag{6.12}$$

$$G_y = \begin{matrix} 1 & 1 \\ -1 & -1 \end{matrix} \tag{6.13}$$

Nyní se gradienty G_x i G_y počítají ve stejném bodě $[i + \frac{1}{2}, j + \frac{1}{2}]$. Tento bod vzniká interpolací, a to může vést k určitým nejasnostem. Proto se dnes častěji využívá oblast 3×3, se kterou pracují také operátory zmíněné v úvodu této kapitoly, a gradient se počítá ve středovém pixelu.

Uveďme nyní Sobelův operátor, který bývá pro detekci hran často využíván. [11][35]

6.3.2. Sobelův operátor [35]

Uvažujme tedy oblast 3×3 pixelů. Pixely popišme následovně:

$$I = \begin{bmatrix} f(x,y) & f(x,y+1) & f(x,y+2) \\ f(x+1,y) & f(x+1,y+1) & f(x+1,y+2) \\ f(x+2,y) & f(x+2,y+1) & f(x+2,y+2) \end{bmatrix},$$

Sobelův operátor je velikost gradientu vypočtená podle vzorce

$$S = \sqrt{S_x^2 + S_y^2}, \tag{6.14}$$

kde se parciální derivace počítají podle vzorce

$$S_x = (f(i,j+2) + cf(i+1,j+2) + f(i+2,j+2)) - (f(i,j) + cf(i+1,j) + f(i+2,j))$$
$$S_y = (f(i,j) + cf(i,j+1) + f(i,j+2)) - (f(i+2,j) + cf(i+2,j+1) + f(i+2,j+2))$$

kde $c = 2$. Stejně jako ostatní gradientní operátory lze i operátory S_x a S_y implementovat pomocí konvolučních masek:

$$S_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \tag{6.15}$$

$$S_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{6.16}$$

Poznamenejme, že tento operátor klade důraz na pixely, které jsou blíže středu masky.

V dalším textu budeme uvažovat oblast o velikosti 3×3 pixely.

6.4. Potlačení nemaximálních hodnot

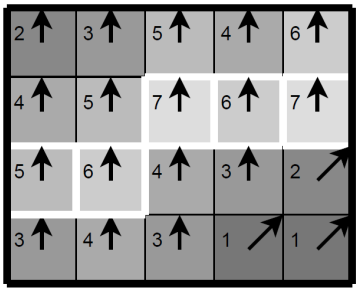
Tento krok zaručí, že hranové pixely se co nejvíce redukují pouze na šířku jednoho pixelu. tedy, že z „rozmazaných hran“ se stanou „ostré“ hrany. V podstatě se zachováme všechny lokální maxima v obraze gradientů a vymažeme všechno ostatní. Algoritmus pro každý pixel je následující:

1. Zaokrouhlení směru gradientu θ na nejbližší násobek 45°, což odpovídá použití spojnic s osmi sousedními (tj. obklopujícími) pixely.

2. Porovnání síly hrany aktuálního pixelu se silou hrany sousedního pixelu v kladném i záporném směru gradientu. Tj. pokud je směr gradientu na sever (tj. $\theta = 90°$), porovnáváme s pixely na sever a na jih.

3. Pokud je síla hrany aktuálního pixelu největší; zachováme hodnotu síly této hrany. Pokud tomu tak není, tuto hodnotu potlačíme (tj. odstraníme).

Příklad potlačení nemaximálních hodnot je na obrázku 6.1. Téměř všechny pixely mají směry gradientu směřující na sever. Jsou proto porovnány s pixely nad a pod nimi. Pixely, které se při tomto porovnání ukáží jako maximální, jsou zobrazeny s bílými okraji. Všechny ostatní pixely budou potlačeny (tj. vynulovány).



Obrázek 6.1: Směry gradientů na hranách. Převzato z [11].

6.5. Dvojité prahování

Pixely, které byly uchovány v předchozím kroku, jsou (stále) označeny svou silou pixel po pixelu. Mnohé z nich budou pravděpodobně skutečnými hranami v obraze, ale některé mohou být dány šumem nebo barevnými odchylkami. Nejjednodušší způsob, jak mezi nimi rozlišit, je použít nějakou formu prahování, takže budou zachovány pouze hrany se silou větší než daná prahovací hodnota.

Cannyho algoritmus detekce hran používá dvojí prahování. Hranové pixely silnější než horní práh jsou označeny jako *silné*; pixely hran slabší než horní práh, ale silnější než spodní prahovací hodnota jsou pak označeny jako *slabé*; nakonec pixely s hodnotami menšími než dolní práh jsou potlačeny.

6.6. Sledování hran pomocí hystereze

V posledním kroku jsou silné hrany interpretovány jako „jisté hrany“ a mohou být ihned zahrnuty do výsledného obrazu s hranami. Slabé hrany jsou zahrnuty pouze tehdy, pokud jsou spojeny se silnými hranami.

Předpoklad zde je, že správně nastavené prahování předchozího kroku zabrání vzniku silných hran ze šumu a jiných odchylek a tyto hrany budou tedy odpovídat těm skutečným. Na druhou stranu slabé hrany mohou být tvořeny buď pravými hranami nebo šumem (resp. barevnými variacemi). Důsledkem toho se velká část z nich nebude vůbec nacházet

v sousedství silných hran. Ty však, které vznikly v důsledku existence skutečné hrany, budou mnohem pravděpodobněji spojeny přímo se silnými hranami.

Pro stopování spojených hranových bodů se využívá takzvaná analýza „blobů“ (ang. blob analysis, česky bychom mohli překládat jako „analýza skvrn“). Lze využít iterativní či rekurzivní přístup.

V iterativním přístupu jsou pixely hran nejprve spojeny do skupin (blobů) detekcí dalších hranových bodu mezi jejich 8 sousedícími pixely. Následně se pro každý bod slabé hrany (opět v okolí 8 pixelů) hledá nějaký bod silné hrany. Pokud je takový bod nalezen jsou slabé hrany daného blobu také označeny za silné a blob je zachován. V opačném případě jsou hrany potlačeny a blob není zahrnut do výsledku. [11]

6.7. Některé nedokonalosti Cannyho algoritmu

Ve srovnání s jinými běžnými algoritmy detekce hran má tradiční Cannyho algoritmus lepší detekční účinek, ale má také některé nedostatky. Autoři [62][63][67] uvádějí zejména následující. Zároveň navrhují vhodná vylepšení.

1. Použití Gaussova filtru na vyhlazení šumu často vede k rozostření obrazu. Tím dochází ke ztrátě slabých hran a zeslabení silných. Mohou také vznikat nové, falešné hrany.
   Autoři článku [62] proto navrhují namísto Gaussova filtru použít adaptivní filtr, který volí váhy v závislosti na skocích signálu a zároveň zostřuje hrany v obraze. Jiné studie k odšumování využívají adaptivní mediánový filtr [67] či statistické filtrování [63].

2. Duální prahování tradičního Cannyho algoritmu sice zajišťuje uchování většího množství hran, je ale obtížné jej správně nastavit. Navíc se prahy nastavují ručně, takže adaptivní schopnost je slabá vzhledem k rozptylu Gaussovy filtrace. To vede ke ztrátě hran nebo nedokonalému odstranění šumu.
   V tomto případě byla jako řešení navrhnuta například Otsuova metoda, která využívá histogram odstínů šedi k výpočtu horního prahu P_h a odvozuje z něj také hodnotu pro dolní práh $P_d = P_h/2$. [67] Další ze způsobů nalezení optimálních prahovacích hodnot je třeba využití tzv. genetického algoritmu [63].

3. Výsledek detekce nemůže ve všech případech dosáhnout velikosti (resp. šířky) pouze jednoho pixelu, ale objevují se i odezvy o velikosti několika bodů. Zvláště body v rozích vytváří odezvu více pixely. Původní Cannyho algoritmus sice dosahuje úrovně uspokojivé pro lidské oko, pro technickou praxi však může být nedostatečná.
   K zúžení detekované hrany byla v článku [62] úspěšně využita matematická morfologie.

6.8. Implementace v programu MATLAB

Protože cílem této diplomové práce není nalézt co nejvíce hran a následně s nimi pracovat dál, ale vybíráme pouze nějaké jejich procento jako referenční hodnoty při rekonstrukci obrazu, budeme jednoduše využívat implementovanou funkci programu MATLAB `edge`.

Dle dokumentace [45] tato funkce poskytuje několik způsobů odhadu derivací, z nichž každý implementuje jednu z těchto dvou definic:

- místa, kde je první derivace intenzity větší než určitá prahová hodnota,
- místa, kde druhá derivace intenzity má průsečík s nulou.

U některých z odhadů lze určit, zda má být operace citlivá na vodorovné hrany, svislé hrany nebo na obojí. Funkce `edge` pak vrací binární obraz s hodnotou 1 v místech, kde jsou nalezeny hrany, a 0 jinde.

Pro nás je podstatné, že nejvýkonnější metodou detekce hran, kterou `edge` poskytuje, je právě Cannyho metoda.

7. Řídké reprezentace signálů

Abychom mohli vůbec pracovat s proximálními algoritmy, na kterých je naše vylepšení JPEG algoritmu postaveno, je potřeba se nejprve alespoň stručně seznámit s řídkými reprezentacemi signálu.

Pokud nebude řečeno jinak, jsou definice a formulace úloh v této kapitole převzaté z [17].

7.1. Základní pojmy

Zadefinujme nejprve některé důležité pojmy.

Definice 7.1. *Nosičem vektoru* $\mathbf{x}$ myslíme množinu jeho indexů, v nichž má vektor nenulové hodnoty. Tuto množinu značíme $\mathrm{supp}(\mathbf{x})$. Tedy $\mathrm{supp}(\mathbf{x}) = \{i | x_i \neq 0\}$.

Definice 7.2. l_p-norma vektoru $\mathbf{x} \in \mathbb{C}^N$ je definována jako

$$\|\mathbf{x}\|_p := |\mathrm{supp}(\mathbf{x})|. \qquad prop = 0 \tag{7.1}$$

$$\|\mathbf{x}\|_p := \sum_{i=1}^{N} |x_i|^p \qquad pro 0 < p < 1, \tag{7.2}$$

$$\|\mathbf{x}\|_p := \left(\sum_{i=1}^{N} |x_i|^p \right)^{\frac{1}{p}} \qquad pro 1 \leq p \leq \infty, \tag{7.3}$$

$$\|\mathbf{x}\|_p := \max_i |x_i|, \qquad prop = \infty \tag{7.4}$$

To že používáme označení l_p-norma pro všechny uvedené případy není matematicky korektní, jde pouze o zjednodušení. Skutečně se jedná o normu pouze v případě $1 \leq p \leq \infty$. $\|\cdot\|_1$ je součtem absolutních hodnot prvků vektoru, $\|\cdot\|_0$ udává počet nenulových složek vektoru a l_2-norma $\|\cdot\|_2$ je pak klasická Euklidovská norma

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{N} |x_i|^2 \right).$$

[57]

Normy v případě matic můžeme definovat několika způsoby. Nejběžněji používanou formou je Frobeniova norma.

Definice 7.3. [57] Nechť $\mathbf{A} \in \mathbb{C}^{m \times N}$. Pak *Frobeniovu normu* definujeme jako

$$\|\mathbf{A}\|_{\mathrm{F}} := \|\mathrm{vec}\mathbf{A}\|_2 = \left(\sum_{i=1}^{N} \sum_{j=1}^{M} |a_{ij}|^2 \right)^{\frac{1}{2}}, \tag{7.5}$$

kde vec$\mathbf{A}$ značí vektorizovanou matici $\mathbf{A}$. $\|\mathbf{A}\|_{\mathrm{F}}$ je také *energií* prvků matice $\mathbf{A}$.

Definice 7.4. Vektor $\mathbf{x} \in \mathbb{C}^N$ nazveme *k-řídkým* (k-sparse), pokud platí

$$\|\mathbf{x}\|_0 \leq k. \tag{7.6}$$

Relativní řídkostí vektoru $\mathbf{x}$ délky N pak budeme rozumět poměr $\frac{k}{N}$. Dále označíme $\sum_k^N := \{\mathbf{x} \in \mathbb{C}^N | \|x\|_0 \leq k\}$ množinu všech k-řídkých vektorů délky N.

Podle této definice je k-řídký takový vektor, který má nejvýše k nenulových složek.[17]

V kapitole o DCT (4) jsme uvedli, že podstatnou informaci o reálném signálu často nese pouze malé množství jeho koeficientů. Mohli bychom tedy takový signál označit za řídký. To by ale neodpovídalo pojetí řídkosti z definice 7.4, protože transformační koeficienty vysokých frekvencí většinou mají velmi malé, avšak ne nenulové velikosti. Proto se používá také pojem *aproximativní řídkost* či *kompresibilita signálu.*[57] Zároveň stále platí, že kompresibilní signály jsou dobře aproximovány těmi řídkými, takže pracovní rámec řídkých aproximací lze aplikovat i na tuto třídu.[64] V [57] autor uvádí „vágní, ale postačující" definici kompresibilního signálu $\mathbf{y} \in \mathbb{R}^n$ jako $\mathbf{y} = \mathbf{A}\mathbf{x} + \epsilon$, kde $\|\epsilon\|_2 \ll \|\mathbf{y}\|_2$ a $\|\mathbf{x}\|_0 \ll n$. To znamená, že signál lze až na malou chybu reprezentovat jako řídkou lineární kombinaci sloupců matice $\mathbf{A}$.

Pokud řekneme, že matice transformačních koeficientů DCT je kompresibilní signál, pak dobrým příkladem řídkého signálu je její kvantizovaná podoba, která obsahuje pouze celočíselné hodnoty. Viz kapitola 5.3.

Mluvíme-li o aproximaci, je vhodné definovat její chybu:

Definice 7.5. *Chyba nejlepší aproximace* vektoru $\mathbf{x} \in \mathbb{C}^N$ k-řídkým vektorem v normě l_p (best k-term approximation error) je definována jako

$$\sigma_k(\mathbf{x})_p := \sigma_k^N(x)_p := \inf_{\mathbf{z} \in \sum_k^N} \|\mathbf{x} - \mathbf{z}\|_p. \tag{7.7}$$

7.2. Řídká řešení systémů lineárních rovnic

Cílem je najít maximálně řídkou reprezentaci pozorovaného signálu $\mathbf{y}$. Matematicky řečeno, hledáme takové řešení standardní nedourčené soustavy lineárních rovnic $\mathbf{A}\mathbf{x} = \mathbf{y}$, že hledaný vektor $\mathbf{x}$ bude co nejřidší. To je, aby obsahoval co největší počet nulových složek. Úlohu formulujeme takto:

$$\arg\min_{\mathbf{x}} \|x\|_0 \quad \text{vzhledem k} \quad \mathbf{A}\mathbf{x} = \mathbf{y}, \tag{7.8}$$

kde vektor $\mathbf{y} \in \mathbb{C}^m$ a matice $\mathbf{A} \in \mathbb{C}^{m \times N}$. Předpokládáme pouze případy, kdy $M < N$, resp. $M \ll N$, a $\mathbf{A}$ je plné (řádkové) hodnosti.

Matici $\mathbf{A}$ obvykle nazýváme *slovník* (dictionary) a její sloupce *atomy* (atoms). Alternativní názvy jsou *reprezentační systém* či poněkud nevhodně *měřící matice* (measurement matrix). Všechna $\mathbf{x}$, která splňují $\mathbf{A}\mathbf{x} = \mathbf{y}$, nazýváme *přípustná řešení*, resp. *přípustné reprezentace vektoru* $\mathbf{y}$. Těchto řešení je nekonečně mnoho a tvoří affiní prostor.[17]

Pokud pozorovaný signál obsahuje šum, dochází při použití předchozího výrazu k odchýlení od přesného řešení. Z toho důvodu povolujeme jistou odchylku δ. Pak má úloha (7.8) podobu:

$$\arg\min_{\mathbf{x}} \|x\|_0 \quad \text{vzhledem k} \quad \|\mathbf{A}\mathbf{x} - \mathbf{y}\|_p \leq \delta, \tag{7.9}$$

kde nejčastěji uvažujeme $p = 2$, tedy Euklidovskou normu.

Prvky (7.9) mohou být různými způsoby kombinovány k vyjádření souvisejících problémů.[64] Například můžeme hledat minimální možnou odchylku vzhledem k danému stupni řídkosti $k \geq 1$

$$\arg\min_{\mathbf{x}} \|\mathbf{Ax} - \mathbf{y}\|_p \quad \text{vzhledem k} \quad \|x\|_0 \leq k. \tag{7.10}$$

Nebo můžeme využít parametr $\lambda > 0$ k vyvažování výsledku jak vzhledem k odchylce, tak i řídkosti

$$\arg\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{Ax} - \mathbf{y}\|_p^2 + \lambda\|\mathbf{x}\|_0. \tag{7.11}$$

Upravená forma tohoto výrazu pro nás bude obzvlášť důležitá v části o proximálním operátoru (8.2).

7.2.1. Postačující podmínky existence řešení

Definice 7.6. Číslo spark($\mathbf{A}$) definujeme jako nejmenší počet sloupců matice $\mathbf{A}$, které jsou lineárně závislé. Zapisujeme

$$\text{spark}(\mathbf{A}) = \arg\min_{\mathbf{x}} \tag{7.12}$$

Pojem spark nemá český ekvivalent, doslova bychom překládali jako „jiskra“. Pro nenulovou matici $\mathbf{A} \in \mathbb{C}^{m\times N}$, kde $m < N$, platí, že spark může nabývat hodnot spark($\mathbf{A}$) $\in \{2, \ldots, m+1\}$, přičemž hodnoty 2 nabývá, když je jeden sloupec násobkem druhého. Čím menší je hodnota spark, tím řidší musí být vektor $\mathbf{x}$, aby byla zajištěna jedinečnost tohoto řešení.[17]

S využitím předchozí definice již můžeme formulovat podmínku existence řešení.

Tvrzení 7.1. *Pokud má soustava* $\mathbf{Ax} = \mathbf{y}$ *řešení* $\mathbf{x}$ *splňující*

$$\|\mathbf{x}\|_0 < \frac{spark(\mathbf{A})}{2}, \tag{7.13}$$

pak $\mathbf{x}$ *je nutně nejřidší možné řešení a žádné jiné řešení se stejnou řídkostí neexistuje.*

Protože je získání spark($\mathbf{A}$) srovnatelně výpočetně náročné jako řešení problému (7.8), potřebujeme jednodušší způsob ověření jedinečnosti řešení. Navíc je předchozí podmínka pouze postačující. To znamená, že mohou existovat nejřidší řešení nesplňující (7.13).[17]

Definice 7.7. Vzájemná koherence (mutual coherence) matice $\mathbf{A}$ je definována jako největší absolutní normalizovaný skalární součin dvou různých sloupců matice $\mathbf{A}$,

$$\mu(\mathbf{A}) = \max_{1\leq j,k\leq N, j\neq k} \frac{|\mathbf{a}_j^T \mathbf{a}_k|}{\|\mathbf{a}_j\|_2 \cdot \|\mathbf{a}_k\|_2}, \tag{7.14}$$

kde $\mathbf{a}_j$ označuje j-tý sloupec matice $\mathbf{A}$.

Tvrzení 7.2. *Pro libovolnou matici* $\mathbf{A}$ *platí*

$$spark(\mathbf{A}) \geq 1 + \frac{1}{\mu(\mathbf{A})}. \tag{7.15}$$

Konečně na základě těchto výrazů můžeme vyjádřit podmínku pro jedinečnost řešení.

Tvrzení 7.3. *Pokud má soustava* $\mathbf{Ax} = \mathbf{y}$ *řešení* $\mathbf{x}$ *splňující*

$$\|x\|_0 < \frac{1}{2}\left(1 + \frac{1}{\mu(\mathbf{A})}\right), \tag{7.16}$$

pak $\mathbf{x}$ *je nutně nejřidší možné a je jediné takové.*

Takového řešení lze dosáhnout l_1-minimalizací. Blíží-li koherence nule, pravá strana nerovnice (7.16) roste nad všechny meze. Pokud se koherence blíží k jedné, také pravá strana jde k jedné. Vzniká tedy snaha používat maximálně nekoherentní slovníky.[17]

7.3. l_1 relaxace

Problém, který nastává v případě normy l_0 je, že tato norma není konvexní. Nemůžeme tedy pro řešení úloh využít žádnou z dnes již rozšířených metod a algoritmů konvexní optimalizace. Lze ale namísto ní využít l_1-normu. Te je její nejbližší konvexní funkcí, takže tato „relaxace“ je vcelku přirozená.[64]

Konvexní forma úlohy (7.8) je

$$\arg\min_{\mathbf{x}} \|x\|_1 \quad \text{vzhledem k} \quad \mathbf{Ax} = \mathbf{y}, \tag{7.17}$$

Pro zašumělá data povolujeme stejně jako v (7.9) jistou odchylku od přesného řešení a úlohu formulujeme ve tvaru

$$\arg\min_{\mathbf{x}} \|x\|_1 \quad \text{vzhledem k} \quad \|\mathbf{Ax} - \mathbf{y}\|_2 \leq \delta. \tag{7.18}$$

Tento problém je také označován zkratkou LASSO (Least Absolute Shrinkage and Selection Operator). l_1-optimalizace může nalézt, v závislosti na použitém algoritmu, i nejřídké řešení. V praxi se ve většině případů řešení úloh (7.9) a (7.18) shodují.

V textu [17] autoři uvádějí dvě podmínky, které zajistí tuto ekvivalenci. Jsou jimi tzv. *vlastnost nulového prostoru* (NSP, null space property) a vlastnost zeslabené izometrie (RIP, restricted isometry property). Podmínka RIP je stabilní i u zašumělých signálů a navíc je oproti NSP výpočetně jednodušší.

8. Proximální algoritmy

V předchozím textu jsem se zabývali optimalizačními úlohami, jejichž řešení vede k řídkým reprezentacím signálů. K nalezení těchto řešení byla vyvinuta celá řada algoritmů a lze řadit do několika logických kategorií. Jmenujme například: hladové algoritmy (greedy algorithms), relaxační algoritmy (založené na konvexní l_1-relaxaci), algoritmy založené na prahování, nekonvexní optimalizaci nebo jednoduše hrubé síle.[64] My se budeme zabývat tzv. *proximálními algoritmy.*

Mnohé úlohy v oblasti zpracování signálů (tzn. také obrazů) mohou být formulovány jako optimalizační problémy, které se skládají z minimalizace sumy konvexních funkcí, ne nezbytně diferencovatelných, případně složených z lineárních operátorů.[12] Uvažujme proto nyní formulaci nějaké obecné tzv. neomezené (unconstrained) konvexní optimalizační úlohy ve tvaru [53]

$$\arg\min_{\mathbf{x}\in\mathbb{R}^n} f\left(\mathbf{x}\right)_1 + \cdots + f\left(\mathbf{x}\right)_m, \tag{8.1}$$

kde $f\left(\mathbf{x}\right)_1 + \cdots + f\left(\mathbf{x}\right)_m : \mathbb{R}^n \to (-\infty, \infty)$.

Proximální algoritmy fungují na principu dělení (splitting) úlohy na jednotlivé funkce $f_1, \ldots, f_m$, které jsou následně používány samostatně. Odtud anglické pojmenování *splitting algorithms.*[53] Označení *proximální* vychází z faktu, že každá nehladká funkce, je zahrnuta skrze její proximální operátor, jehož definici uvedeme v dalších podkapitolách. Pro odvození algoritmu, se kterým budeme později pracovat, nám nyní postačí omezit se dle [57] na úlohu o dvou členech

$$\arg\min_{\mathbf{x}} f_1(\mathbf{x}) + f_2(\mathbf{x}). \tag{8.2}$$

Na funkce f_1, f_2 jsou kladeny jisté požadavky, které ale bude vhodnější uvést až později v textu.

8.1. Optimalizační úloha v omezeném a neomezeném tvaru

Úloha pro l_1-normu (7.17) je optimalizační úlohou v *omezeném* tvaru. Připomeňme, že úloha byla formulována jako:

$$\arg\min_{\mathbf{x}} \|\mathbf{x}\|_1 \quad \text{vzhledem k} \mathbf{Ax} = \mathbf{y}. \tag{8.3}$$

Tato forma sice neodpovídá neomezené (8.2), kterou chceme řešit, ale s využitím indikátorové funkce ji lze na požadovaný tvar převést.

Definice 8.1 (Indikátorová funkce [57]). Nechť $\mathbf{C}$ je neprázdná podmnožina $\mathbb{R}^n$. *Indikátorová (charakteristická) funkce* množiny $\mathbf{C}$ je

$$\iota_C \;:\; \mathbf{x} \mapsto \begin{cases} 0 & \text{pro } \mathbf{x} \in \mathbf{C} \\ \infty & \text{jinak.} \end{cases} \tag{8.4}$$

Nová úloha nyní bude

$$\arg\min_{\mathbf{x}} \|\mathbf{x}\|_1 + \iota_{\{\mathbf{x}:\ \mathbf{Ax}=\mathbf{y}\}}, \tag{8.5}$$

přičemž druhý sčítanec zároveň vynucuje náležitost řešení do přípustné množiny.[57]

Podobně omezenou úlohu(7.18)

$$\arg\min_{\mathbf{x}} \|\mathbf{x}\|_1 \text{ vzhledem k } \|\mathbf{Ax}-\mathbf{y}\|_2 \leq \delta, \tag{8.6}$$

jsme schopni převést na

$$\arg\min_{\mathbf{x}} \|\mathbf{x}\|_1 + \iota_{\{\mathbf{x}:\ \|\mathbf{Ax}-\mathbf{y}\|_2 \leq \delta\}}. \tag{8.7}$$

Dále je také možné (a pro naši další práci nezbytné) vyjádřit (8.6) jako tzv. *regularizovanou lineární inverzní úlohu* [57]:

$$\arg\min_{\mathbf{x}} \frac{1}{2}\|\mathbf{Ax}-\mathbf{y}\|_2^2 + \lambda\|x\|_1. \tag{8.8}$$

Tato neomezená úloha je ekvivalentní s omezenou úlohou (8.6) ve smyslu: ke každému δ lze nalézt $\lambda > 0$, tak, že obě úlohy dávají stejný vektor optimálních $\mathbf{x}$.[57] Mimo jiné odpovídá i smíšené formuli (7.11).

Člen $\lambda \geq 0$ je regularizační parametr, jehož volbou ovlivňujme řídkost řešení. Jinými slovy, penalizujeme neřídká řešení. Čím vyšší je λ, tím řidší dostáváme výsledky. Příliš vysoké hodnoty však mohou způsobit odchylku od dat. Pro samotnou volbu λ neexistuje žádné doporučení a vybrat správnou hodnotu předem může být složité, protože reguluje řídkost nepřímo. V důsledku tak často musíme řešit (8.8)opakovaně pro různé volby λ nebo systematicky sledovat vývoj řešení jak se λ přibližuje k 0. Když $\lambda \geq \|^*\mathbf{y}\|_\infty$, řešení (8.8) je $\mathbf{x} = \mathbf{0}$.[57][64]

8.2. Proximální operátor

V 60. letech 20. století navrhl Jean Jacques Moreau užitečné rozšíření definice projekčního operátor, který hledá bod v této množině ležící nejblíže výchozímu bodu, na jakoukoliv obecnou konvexní funkci. Toto rozšíření vede k takzvanému *proximálnímu operátoru.*[21] Před tím než definujeme samotný operátor, je nutné zadefinovat několik potřebných pojmů.

Definice 8.2 (Zdola polospojitá funkce). [57] Řekneme, že funkce $f(x)$ je zdola polospojitá na podmnožině $\mathbf{U}$ metrického prostoru, jestliže pro libovolné $\alpha \in \mathbb{R}$ je množina $\{\mathbf{x} : f(\mathbf{x}) > \alpha\}$ otevřená.

Definice 8.3. [53] Jako $\Gamma_0(\mathbb{R}^n)$ označíme třídu zdola polospojitých konvexních funkcí $f : \mathbb{R}^n \to (-\infty, \infty)$ takových, že $\mathrm{dom} f \neq \emptyset$ (tj. mají neprázdný definiční obor).

Definice 8.4 (Subdiferenciál). [57] Nechť $f \in \Gamma_0(\mathbb{R}^n)$. *Subdiferenciál* funkce f je operátor $\partial f : \mathbb{R}^n \to 2^{\mathbb{R}^n}$, který definujeme jako

$$\mathbf{x} \mapsto \left\{\mathbf{u} \in \mathbb{R}^n : (\forall \tilde{\mathbf{x}} \in \mathbb{R}^n)(\tilde{\mathbf{x}} - \mathbf{x})^T\mathbf{u} + f(\mathbf{x}) \leq f(\tilde{\mathbf{x}})\right\} \tag{8.9}$$

Jednotlivé prvky $\mathbf{u}$ subdiferenciálu se nazývají *subgradienty.*

Subgradient ∂f je zobecněný pojem ke gradientu $\nabla f = \left[\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}\right]$ pro nediferencovatelné funkce. V bodech, kde f je diferencovatelná tyto pojmy splývají. Subgradient ∂f je operátor, který přiřazuje funkci v bodě množinu.[57]

Také definujme projekci, zmíněnou v úvodu.

Definice 8.5. [53] Projekce $P_C\mathbf{x}, \mathbf{x} \in \mathbb{R}^n$ na neprázdnou uzavřenou konvexní množinu $C \subset \mathbb{R}^n$ je řešením problému

$$\arg\min_{\mathbf{y}\in\mathbb{R}^n} \iota_C(\mathbf{y}) + \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|^2, \tag{8.10}$$

kde funkce ι_C náleží do $\Gamma_0(\mathbb{R}^n)$.

Moreaův návrh rozšíření projekčního operátoru v [50] spočívá v nahrazení ι_C ve výrazu (8.10) libovolnou funkcí $f \in \Gamma_0(\mathbb{R}^n)$.

Definice 8.6 (Proximální operátor). Nechť $f \in \Gamma_0(\mathbb{R}^n)$. Pro každé $\mathbf{x} \in \mathbb{R}^n$ má minimalizační problém

$$\arg\min_{\mathbf{y}\in\mathbb{R}^n} \frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 + f(\mathbf{y}). \tag{8.11}$$

jednoznačné řešení, které budeme značit $\mathrm{prox}_f(x)$. Operátor $\mathrm{prox}_f() : \mathbb{R}^n \to \mathbb{R}^n$ se nazývá *proximální operátor* funkce f.

Autor v [57] vhodně poznamenává, že výraz (8.11) lze chápat jako regularizované potlačování šumu, přičemž člen $\|\mathbf{x}-\mathbf{y}\|_2^2$ zajišťuje, že výsledek odšumování nebude „příliš daleko“ od známého signálu $\mathbf{x}$. Dále, člen $f(\mathbf{y})$ zastupuje roli regularizačního parametru. To znamená, že penalizuje hledané $\mathbf{y}$ pomocí matematicky formulované konvexní funkce a tím vynucuje některé vlastnosti $\mathbf{y}$, jako například: energii $f(\mathbf{y}) = \|\mathbf{y}\|_2$, relaxovanou řídkost $f(\mathbf{y}) = \|\mathbf{y}\|_1$ nebo totální variaci $f(\mathbf{y}) = \|\mathbf{y}\|_{\mathrm{TV}}$.
Proximální operátor funkce $f \in \Gamma_0(\mathbb{R}^n)$ lez charakterizovat také inkluzí [53]

$$\forall \mathbf{x}, \mathbf{p} \in \mathbb{R}^n \times \mathbb{R}^n : \mathbf{p} = \mathrm{prox}_f(\mathbf{x}) \Leftrightarrow \mathbf{x} - \mathbf{p} \in \partial f(\mathbf{p}). \tag{8.12}$$

Pokud je f navíc diferencovatelná, pak lze výraz redukovat na

$$\forall \mathbf{x}, \mathbf{p} \in \mathbb{R}^n \times \mathbb{R}^n : \mathbf{p} = \mathrm{prox}_f(\mathbf{x}) \Leftrightarrow \mathbf{x} - \mathbf{p} \in \nabla f(\mathbf{p}). \tag{8.13}$$

8.2.1. Vlastnosti proximálního operátoru

Proximální operátory mají některé charakteristiky, které z nich dělají vhodné nástroje pro iterativní minimalizační algoritmy. Například, $\mathrm{prox}_f()$ je silně neexpanzivní, tj.

$$\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^n : \left\|\mathrm{prox}_f(\mathbf{x}) - \mathrm{prox}_f(\mathbf{y})\right\|^2 + \left\|(\mathbf{x} - \mathrm{prox}_f(\mathbf{x})) - (\mathbf{y} - \mathrm{prox}_f(\mathbf{y}))\right\|^2 \leq \|\mathbf{x}-\mathbf{y}\|^2$$

a jeho množina pevných bodů je právě množina minimalizátorů f.
Takovéto vlastnosti nám umožňují vytvářet algoritmy založené na proximálních operátorech $(\mathrm{prox}_{f_i}())_{1\leq i\leq m}$ k vyřešení (8.2) (resp. (8.1)). Přičemž tyto algoritmy do jisté míry napodobují způsob jakým konvexní algoritmy využívají operátor projekce $(P_{C_i})_{1\leq i\leq m}$ k vyřešení úlohy

$$x_{n+1} = P_{C_1} \cdots P_{C_m} x_n,$$

Vlastnost	$f(x)$	$\text{prox}_f(x)$
translace	$\varphi(x-z), z \in \mathbb{R}^n$	$z + \text{prox}_\varphi(x-z)$
změna měřítka	$\varphi(x/\rho)$	$\rho\,\text{prox}_{\frac{\varphi}{\rho^2}}(x/\rho)$
reflexivita	$\varphi(-x)$	$-\text{prox}_\varphi(-x)$
sdružená funkce	$\varphi^*(x)$	$x - \text{prox}_\varphi(x)$
indikátorová funkce	$\iota_C(x)$	$P_C x$

Tabulka 8.1: Stručný přehled vybraných vlastností proximálního operátoru.

kde $C \in \mathbb{R}^n$ a $C \neq \emptyset$.[53]

I když máme v rukách skutečně dobrý nástroje minimalizace, jeho použití ztěžuje fakt, že pro většinu funkcí f není triviální záležitostí proximální operátor najít; pro některé f ani neexistuje explicitní zápis a je nutné je aproximovat iterativně. Také naopak, existují heuristické shrinkage operátory, jejichž explicitní vyjádření regularizačního členu f v (8.11) neznáme nebo ani neexistují.[57]
V tabulce 8.1 jsou uvedené některé z nejzákladnějších vlastností proximálního operátoru. Podrobnější přehledy vlastností a odvozených proximálních operátorů lze nalézt například v [53] nebo [21].

8.2.2. Proximální operátor l_1-normy

Jak už bylo naznačeno v kapitole 7 o řídkých reprezentacích, l_1-norma je při hledání řídkých reprezentací signálů a v komprimovaném snímání zcela klíčová.
Pro případ $f(\mathbf{y}) = \lambda\|\mathbf{y}\|_1$, kde λ je regularizační skalár dostáváme

$$\text{prox}_{\lambda\|\dots\|_1}(\mathbf{x}) = \underset{\mathbf{y}\in\mathbb{R}^n}{\arg\min}\,\frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{y}\|_1. \tag{8.14}$$

rovnici lze rozepsat do tvaru

$$\underset{\mathbf{y}\in\mathbb{R}^n}{\arg\min}\,\frac{1}{2}\sum_{i=1}^{n}(x_i-y_i)^2 + \lambda\sum_{i=1}^{n}\|y_i\|. \tag{8.15}$$

Pro všechna $x_i \neq 0$ lze odvodit funkci

$$y_i\frac{x_i}{\|x_i\|}\max(\|x_i\|-\lambda, 0). \tag{8.16}$$

Nazýváme ji *měkké prahování* (soft tresholding) a značíme $y_i = \text{soft}_\lambda(x_i)$. Důležité je, že každá složka vektoru $\mathbf{x}$ je prahována zvlášť.[57]

8.2.3. Proximální operátor totální variace

Pro 1D totální variaci má proximální operátor tvar

$$\text{prox}_{\lambda\|\dots\|_{\text{TV}}}(\mathbf{x}) = \underset{\mathbf{y}\in\mathbb{R}^n}{\arg\min}\,\frac{1}{2}\|\mathbf{x}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{y}\|_{\text{TV}}. \tag{8.17}$$

Výraz lze rozepsat jako

$$\underset{\mathbf{y}\in\mathbb{R}^n}{\arg\min}\,\frac{1}{2}\sum_{i=1}^{n}(x_i-y_i)^2 + \lambda\sum_{i=1}^{n-1}\|y_{i+1}-y_i\|. \tag{8.18}$$

Poznamenejme, že TV-norma je v podstatě l_1-normou diferencí signálu. V optimalizačních úlohách TV-norma směřuje k signálům, jež mají řídký gradient, tedy takovým, které jsou po částech konstantní![57]

8.3. Proximální algoritmus

Existuje mnoho variant proximálních algoritmů. Stručný výčet i s popisem lze nalézt např. v [53]. Tyto algoritmy se navzájem liší podle operátorové funkce, kterou jsou schopné minimalizovat. Je dobré také připomenout, že tyto algoritmy nejsou schopné nalézt přesné řešení, ale pouze jeho aproximaci, jak ostatně i název napovídá. Mezi klasické algoritmy patří dopředně-zpětné dělní a Douglas-Rachfordův, přičemž oba minimalizují sumu dvou funkcí. Popišme nyní ve stručnosti první z algoritmů a ukažme tak zároveň princip na jakém tyto algoritmy fungují.

8.3.1. Algoritmus dopředně-zpětného dělení

Vraťme se nyní k neomezené úloze (8.2) a přeznačme ji (především z praktických důvodů) na

$$\arg\min_{\mathbf{x}} g(\mathbf{x}) + h(\mathbf{x}), \tag{8.19}$$

kde $h : \mathbb{R}^n \to \mathbb{R}$ je zdola polospojitá konvexní funkce a $g : \mathbb{R}^n \to \mathbb{R}$ je konvexní funkce, která je diferencovatelná s β-lipschitzovským spojitým gradientem ∇g. To znamená, že pro každé $[\mathbf{x}, \mathbf{y}] \in \mathbb{R}^n \times \mathbb{R}^n$ platí

$$\|\nabla g(\mathbf{x}) - \nabla g(\mathbf{y})\| \leq \beta \|\mathbf{x} - \mathbf{y}\|, \tag{8.20}$$

kde $\beta \in (0, \infty)$. Tato úloha má alespoň jedno řešení, pokud $g + h$ je koercivní, tj. $\lim_{\|\mathbf{x}\|\to\infty} g(\mathbf{x}) + h(\mathbf{x}) = \infty$. Řešení bude jediné, pokud platí, že $g + h$ je na také ryze konvexní (tj. alespoň jedna z funkcí g, h je ryze konvexní).[57] Pro $\mathbf{x} \in \mathbb{R}^n$ a pro všechna $t \in (0, \infty)$ platí, že $\mathbf{x}$ je řešením úlohy právě tehdy, když

$$\mathbf{x} = \mathrm{prox}_{th}\left(\mathbf{x} - t \cdot \nabla g(\mathbf{x})\right). \tag{8.21}$$

Tato rovnice implikuje možnost numerického řešení (8.19). Iterováním

$$\mathbf{x}^{k+1} = \mathrm{prox}_{t^k h}\left(\mathbf{x}^k - t^k \cdot \nabla g(\mathbf{x}^k)\right) \tag{8.22}$$

pro $t^k < \frac{2}{\beta}$, kde k je číslo iterace, konverguje posloupnost $\mathbf{x}^k$ k řešení problému (8.19). Tento algoritmus nazýváme *proximální gradientní metoda* nebo *dopředné-zpětné dělení* (forward-backward splitting). Jako dopředný krok chápeme část $\mathbf{x}^k - t^k \cdot \nabla g(\mathbf{x}^k)$, která na výpočet mezikroku $\mathbf{x}^{k+\frac{1}{2}}$ využívá pouze funkci g. Zpětný krok tvoří proximální operátor $\mathrm{prox}_{t^k h}$, který počítá $\mathbf{x}^{k+1}$ zase jenom s využitím h.[53][57]

Algoritmus 1 (Proximální gradientní algoritmus).

1. *Zvolíme* $\epsilon \in (0 \min\{1, 1/\beta\})$.
2. *Zvolíme počáteční bod* $\mathbf{x}_0 \in \mathbb{R}^n$.
3. *Pro* $k = 0, 1, \ldots :$

- *Zvolíme* $t^k \in [\epsilon, 2/\beta - \epsilon]$.
- *Vypočítáme* $x^{k+1/2} = x^k - t^k \cdot \nabla g(x^k)$.
- *Zvolíme* $\lambda^k \in [\epsilon, 1]$.
- *Vypočítáme* $x^{k+1} = x^k + \lambda^k(prox_{t^k h} x^{k+1/2} - x^k)$.

8.3.2. Obecný proximální algoritmus podle Condata

Condat v [12] navrhuje obecnou formulaci úlohy (8.1) a na jejím základě univerzální proximální algoritmus.

Obecný optimalizační problém Condat definuje pro reálné Hilbertovy prostory $\mathcal{X}$ a $\{\mathcal{U}_m\}_{m=1}^M$ a pro nějaké $M \in \mathbb{N}$. Hledáme $\hat{x}$ takové, že

$$\hat{x} \in \arg\min_{x \in \mathcal{X}} f(x) + g(x) + \sum_{m=1}^{M} h(L_m x)_m, \tag{8.23}$$

kde:

- $f, g \in \Gamma_0(\mathcal{X}), h_m \in \Gamma_0(\mathcal{U}_m)$, přičemž označení Γ_0 odpovídá naší definici (8.3) s aplikací na prostory $\mathcal{X}, \mathcal{U}_m$ místo $\mathbb{R}^n$.
- Operátory $L_m : \mathcal{X} \mapsto \mathcal{U}_m$ jsou lineární a ohraničené.
- f je diferencovatelná na $\mathcal{X}$ a její gradient ∇f je β-lipschitzovsky spojitý (viz vztah (8.20)).
- Je splněno omezení:

  $$(0, \ldots, 0) \in \mathrm{sri}\{(L_m x - u_m)_{1 \le m \le M} | x \in \mathrm{dom}(g) \text{ a } u_m \in \mathrm{dom}(h_m), \forall m = 1, \ldots, M\}$$

  , kde sri značí silně relativní vnitřek množiny.

V tomto obecném vyjádření je možné, aby některé funkce byly nulové a za operátor L_m lze dosadit identitu I. Toho také později využijeme.

Prostory $\mathcal{X}$ a $\mathcal{U}_m$ mají často velké dimenze N, což následně vede na náročnou optimalizaci. Navíc ve většině případů ani není technicky možné v každé iteraci algoritmu manipulovat s rozsáhlými maticemi (např. $N = 10^6$). Z hlediska výpočetní i teoretické náročnosti je proto lepší řešit problém „úplným rozdělením“. To znamená, že v každé iteraci jediné prováděné operace budou výpočty $\nabla f, \mathrm{prox}_g(\cdot), \mathrm{prox}_{h_m}(\cdot), L_m$ nebo adjungovaného operátoru L_m^*. Proto je vyžadováno, aby tyto výpočty byly dostatečně jednoduché. To znamená, aby mohly být provedeny za čas $O(N)$ nebo $O(N \log(N))$, kde N je velikost okolního prostoru $\mathcal{X}$ nebo $\mathcal{U}_m$.[12]

V algoritmu se pro proximální operátory objevují také funkce h_m^*. Jde o konvexně sdružené funkce a definujeme je dle [58] takto

Definice 8.7 (Konvexně sdružená funkce). Nechť $\mathcal{X}$ je Hilbertův prostor se skalárním součinem $\langle \cdot, \cdot \rangle$ a $f : H \to \mathbb{R} \cup \{-\infty, \infty\}$. Funkci konvexně sdruženou k funkci f značíme f^* a definujeme

$$f^*(x) = \sup_{u \in H} \langle \cdot, \cdot \rangle - f(u). \tag{8.24}$$

Algoritmus řešící úlohu (8.23) je následující:

Algoritmus 2 (Obecný proximální algoritmus).

1. *Zvolíme* $\tau > 0, \sigma > 0$ *a* $\rho > 0$.
2. *Zvolíme počáteční odhady* $x^{(0)} \in \mathcal{X}, u_1^{(0)} \in \mathcal{U}_1, \ldots, u_1^{(M)} \in \mathcal{U}_M$.
3. *Iterujme pro* $i = 0, 1, \ldots$:
   - *Vypočítáme* $\tilde{x}^{(i+1)} := prox_{\tau g}\left(x^{(i)} - \tau \nabla f\left(x^{(i)}\right) - \tau \sum_{m=1}^{M} L_m^* u_m^{(i)}\right)$.
   - *Vypočítáme* $x^{(x+1)} := \rho \tilde{x}^{(i+1)} + (1-\rho)\, x^{(i)}$.
   - *Pro* $m = 1, \ldots, M$:
     - *Vypočítáme* $\tilde{u}_m^{(i+1)} := prox_{\sigma h_m^*}\left(u_m^{(i)} + \sigma L_m \left(2\tilde{x}^{(i+1)} - x^{(i)}\right)\right)$.
     - *Vypočítáme* $u_m^{(i+1)} := \rho \tilde{u}_m^{(i+1)} + (1-\rho) u_m^{(i)}$.

Condat navrhuje dvě verze algoritmu, které se od sebe prakticky liší pouze pořadím kroků. Rozdíl vzniká až při implementaci, kdy v závislosti na typu problém může interpolační krok $2\tilde{\circ}^{(i+1)} - \circ^{(i)}$ vést k různým nárokům na operační paměť.

Poznamenejme, že v případě $f = 0$ se navrhovaný algoritmus podobá Chambolle-Pockovu algoritmu [19] s relaxací navíc. Pokud $M = 0$ a minimalizujeme pouze $f(x) + g(x)$, pak algoritmus odpovídá proximálnímu gradientnímu (algoritmus 2).

8.3.3. Konvergence Condatova algoritmu

O konvergenci předchozího algoritmu mluví následující dvě tvrzení.

Tvrzení 8.1. *Předpokládejme, že parametry v algoritmu 2 splňují podmínky:*

1. $\tau\left(\frac{\beta}{2} + \sigma\left\|\sum_{m=1}^{M} L_m^* L_m\right\|\right) < 1$, *kde Lipschitzovská konstanta* β *je definovaná výrazem (8.20) a* $\|\cdot\|$ *je operátorová norma,*
2. $\rho \in (0, 2)$.

Potom obě posloupnosti $\left(\tilde{x}^{(i)}\right)_{i\in\mathbb{N}}$ *a* $\left(x^{(i)}\right)_{i\in\mathbb{N}}$ *generované algoritmem 2 konvergují (slabě, pokud* X *má nekonečnou dimenzi) k* $\hat{x} \in X$ *řešení problému (8.23).*

Tvrzení 8.2. *Předpokládejme, že* $f = 0$, *prostory* $\mathcal{X}$ *a* $\mathcal{U}_{\Updownarrow}$ *mají konečné dimenze a že parametry algoritmu 2 splňují podmínky:*

1. $\tau\sigma\left\|\sum_{m=1}^{M} L_m^* L_m\right\| \leq 1$.
2. $\rho \in (0, 2)$.

Potom obě posloupnosti $\left(\tilde{x}^{(i)}\right)_{i\in\mathbb{N}}$ *a* $\left(x^{(i)}\right)_{i\in\mathbb{N}}$ *generované algoritmem 2 konvergují k řešení* $\hat{x} \in X$ *problému (8.23).*

Podle 8.2 dovolujeme parametru ρ se blížit k hodnotě 2, což může výrazně urychlit konvergenci. Navíc je konvergence zaručena volbou $\sigma\tau\|\sum_m L_m^* L_m\| = 1$, což autor článku doporučuje využít i v praxi. Tato nová podmínka umožňuje použít Douglasův-Rachfordův algoritmus jako speciální případ algoritmu 2, kde $f = 0, M = 1, L_1 = \mathrm{Id}$ a nastavením $\sigma = \frac{1}{\tau}$ se τ bude tvářit jako jediný parametr.

9. Odvození proximálního algoritmu

Optimalizační úloha a proximální algoritmus, které v následujícím textu odvodíme a vysvětlíme, budou nepostradatelnou částí našeho nového JPEG dekodéru. Hlavní myšlenka spočívá v uchování dodatečné, nezměněné informace o obrazu během procesu kódování obrazu (viz oddíl 5.3) a její následné použití k přesnější rekonstrukci komprimovaného obrazu.

9.1. Optimalizační úloha

Optimalizační úlohu formulujeme na základě Condatovy definice obecného optimalizačního problému (8.23). V našem případě volíme $M = 2$. Obecnou formulaci problému dostáváme ve tvaru

$$\arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) + g(\mathbf{x}) + \sum_{m=1}^{M=2} h\left(L_m\mathbf{x}\right)_m .$$

Což triviálním výpočtem rozepíšeme jako

$$\arg\min_{\mathbf{x}\in\mathcal{X}} f(\mathbf{x}) + g(\mathbf{x}) + h\left(L_1\mathbf{x}\right)_1 + h\left(L_2\mathbf{x}\right)_2 . \tag{9.1}$$

Pro funkce f, g, h_1, h_2 a operátory L_1, L_2 platí stejné podmínky, jaké byly uvedeny pro úlohu (8.23).

Úloha minimalizuje hodnoty $\mathbf{x}$, což jsou již z binární reprezentace dekódované a dekvantizované transformační koeficienty. Jednotlivé funkce a operátory volíme takto:

Funkci $f(x)$ volíme nulovou, protože v našem případě nemáme žádnou diferencovatelnou funkci, kterou bychom použili.

Člen $g(x)$ zajistí, že koeficienty upravené v běhu algoritmu nebudou příliš vzdálené od těch původních. Označme c_k koeficienty, které algoritmus upravuje, a c_k^Q původní koeficienty, které vstupují do algoritmu (resp. nyní do úlohy(9.1)) na počátku. Protože úlohu řešíme pro bloky 8×8 (viz část 5.3), index $k = 0, \ldots, 63$. Je důležité zdůraznit, že jak koeficienty c_k tak i c_k^Q jsou dekvantizované! My však požadujeme, aby od sebe nebyly příliš vzdálené v kvantizované oblasti. Musíme je proto opět vydělit odpovídajícími koeficienty Q_k kvantizační matice $\mathbf{Q}$.

Chceme, aby koeficienty c_k a c_k^Q od sebe nebyly dále než $\frac{1}{2}$ bodu. Máme tedy podmínku:

$$-\frac{1}{2} \leq \frac{c_k - c_k^Q}{Q_k} \leq \frac{1}{2} - \frac{Q_k}{2} \leq c_k - c_k^Q \leq \frac{Q_k}{2} |c_k - c_k^Q| \leq \frac{Q_k}{2}$$

Funkci $g(x)$ pak vyjádříme jako indikátorovou funkci s uvedenou podmínkou:

$$g(\mathbf{x}) = \iota_{\{\mathbf{c}:|c_k - c_k^Q| \leq \frac{\mathbf{Q}_k}{2},\, k=0,\ldots,63\}}(\mathbf{x}) .$$

Takto umožníme korekci obrazu, ale zároveň zabráníme tomu, aby se koeficienty příliš změnily. Připomeňme, že tyto dekvantizované koeficienty reprezentují frekvence v obrazu. Pokud bychom je příliš změnili, mohli bychom v obrazu vytvořit nové viditelné chyby nebo při velkých odchylkách obraz zcela zdeformovat.

Členy $h\left(L_m\mathbf{x}\right)_m$ jsou zřejmě složením funkce a operátoru $h_m \circ L_m$. Po obou funkcích h_1 a h_2 budeme chtít, aby upravovaly rekonstruovaný obraz v prostorové oblasti. Jak už ale bylo řečeno, naše vstupní hodnoty $\mathbf{x}$ jsou dekvantizovanými koeficienty a proto je nejdříve do prostorové oblasti musíme převést. To uděláme pomocí operátorů L_1, L_2, které pro nás budou inverzními DCT:

$$L_1 = \mathrm{IDCT}\left(\mathbf{x}\right), \tag{9.2}$$

$$L_2 = \mathrm{IDCT}\left(\mathbf{x}\right). \tag{9.3}$$

Ve skutečnosti však bude $L_1 = \mathrm{D} \circ \mathrm{IDCT}\left(\mathbf{x}\right)$. Proč tomu tak je, je vysvětleno níže.

Pro volbu funkce h_1 budeme nejprve potřebovat zadefinovat diskrétní totální variaci. V [12] je TV definována tímto způsobem.

Definice 9.1 (Diskrétní totální variace). Nechť $\mathrm{D} : \mathcal{X} \to \mathcal{X}^2$ je diskrétní diferenční operátor, který mapuje obraz $\mathbf{x}$ na dvojici obrazů $(\mathbf{u}_h, \mathbf{u}_v)$ tak, že pro každé $k_h = 1, \ldots, N_h, k_v = 1, \ldots, N_v$ platí

$$u_h\left[k_h, k_v\right] = \left\{x\left[k_h, k_v\right] - \left[k_h - 1, k_v\right] \text{ pokud } k_h \geq 2, 0 \text{ jinak}\right\}, \tag{9.4}$$

$$u_v\left[k_h, k_v\right] = \left\{x\left[k_h, k_v\right] - \left[k_h, k_v - 1\right] \text{ pokud } k_v \geq 2, 0 \text{ jinak}\right\}. \tag{9.5}$$

Pak *diskrétní totální variaci* $\mathrm{TV}(\mathbf{x})$ definujeme jako

$$\mathrm{TV}(\mathbf{x}) = \left\|\mathrm{D}\mathbf{x}\right\|_{1,2}, \tag{9.6}$$

kde

$$\left\|(\mathbf{u}_h, \mathbf{u}_v)\right\|_{1,2} = \sum_{k_h=1}^{N_h} \sum_{k_v=1}^{N_v} \sqrt{u_v\left[k_h, k_v\right]^2 + u_h\left[k_h, k_v\right]^2}.$$

Abychom byli schopní později vyjádřit proximální operátor funkce h_1 nemůžeme použít přímo výraz (9.6), ale podobně jako v [12] položíme $h_1 = \left\|L_1\right\|_{1,2}$ a diferenční operátor D zahrneme do operátoru L_1, takže $L_1 = \mathrm{D} \circ \mathrm{IDCT}\left(\mathbf{x}\right)$.

Význam využití totální variace (ve zpracování signálu obecně) spočívá v její schopnosti redukovat nechtěný šum v obraze.

Zbývá volba funkce h_1. Jedním ze vstupů do našeho algoritmu budou také uložené hranové pixely původního obrázku. Označme je jako matici $\mathbf{M}_e$. Chceme, aby tyto pixely zůstaly během rekonstrukce obrazu nezměněné a algoritmus tak na jejich základě v každé iteraci zpřesňoval aproximaci okolních bodů. Díky tomu, že ukládáme záměrně hodnoty na hranách, bude vliv této korekce největší v místech, kde obvykle bývají nechtěné blokové artefakty nejvýraznější.

Funkci h_2 vyjádříme jako indikátorovou funkci ve tvaru

$$h_2 = \iota_{\{\mathbf{y}:\mathbf{M}_y = \mathbf{M}_e\}},$$

kde $\mathbf{M}_y$ je matice hodnot pixelů aproximovaných v dané iteraci algoritmu.

Vzhledem k výše popsané volbě funkcí a operátorů můžeme nyní dosadit do obecné formulace (9.1). Výsledný tvar úlohy je

$$\arg\min_{\mathbf{x}\in\mathcal{X}} \iota_{\{\mathbf{c}:|c_k - c_k^Q| \leq \frac{\Delta k}{2},\, k=0,\ldots,63\}}(\mathbf{x}) + \left(\mathrm{D}\left(\mathrm{IDCT}\left(\mathbf{x}\right)\right)\right) + \iota_{\{\mathbf{y}:\mathbf{M}_y = \mathbf{M}_e\}}(\mathrm{IDCT}\left(\mathbf{x}\right)). \tag{9.7}$$

9.2. Odvozený algoritmus

Uveďme nyní algoritmus, řešící úlohu (9.7), přičemž proximální operátory, volba parametrů a ukončovací podmínka jsou uvedeny níže.

Algoritmus 3 (Upravený proximální algoritmus).

1. *Nastavíme vstupní hodnoty x_0 jako dekvantizované koeficienty.*
2. *Nastavíme počáteční hodnoty $u_{1,0}, u_{2,0}$ jako nulové matice.*
3. *Dokud platí podmínka (9.16), pak pro $i = 0, 1, \ldots$:*
   - $\tilde{x}^{(i+1)} := prox_{\tau g}\left(x^{(i)} - \tau(L_1^* u_1^{(i)} + L_2^* u_2^{(i)})\right)$
   - $x^{(i+1)} := \rho\tilde{x}^{(i+1)} + (1-\rho)x^{(i)}$
   - *Pro $m = 1, \ldots, M$, kde $M = 2$:*
     - $\tilde{u}_1^{(i+1)} := prox_{\sigma h_1^*}\left(u_1^{(i)} + \sigma L_1(2\tilde{x}^{(i+1)} - x^{(i)})\right) =$
     $$= prox_{\sigma h_1^*}\left(u_1^{(i)} + \sigma D\left(IDCT\left(2\tilde{x}^{(i+1)} - x^{(i)}\right)\right)\right)$$
     - $u_1^{(i+1)} := \rho\tilde{u}_1^{(i+1)} + (1-\rho)u_1^{(i)}$
     - $\tilde{u}_2^{(i+1)} := prox_{\sigma h_2^*}\left(u_2^{(i)} + \sigma L_2(2\tilde{x}^{(i+1)} - x^{(i)})\right) =$
     $$= prox_{\sigma h_2^*}\left(u_2^{(i)} + \sigma IDCT\left(2\tilde{x}^{(i+1)} - x^{(i)}\right)\right)$$
     - $u_2^{(i+1)} := \rho\tilde{u}_2^{(i+1)} + (1-\rho)u_2^{(i)}$

Pro adjungované operátory L_1^*, L_2^* k operátorům L_1, L_2 platí:

$$L_1^* = (\mathrm{D}(\cdot) \circ \mathrm{IDCT}(\cdot))^* = \mathrm{IDCT}^*(\cdot) \circ \mathrm{D}^*(\cdot) = \mathrm{DCT}(\cdot) \circ \mathrm{D}^*(\cdot), \tag{9.8}$$

$$L_2^* = \mathrm{DCT}(\cdot) \tag{9.9}$$

9.2.1. Proximální operátory

Proximální operátory pro funkce g, h_1, h_2 z úlohy (9.1) (resp. jejich formulace v (9.7)), které jsme použili v uvedeném algoritmu, definujeme následujícím způsobem.

Pro funkci g definujeme $\mathrm{prox}_{\tau g}(u)$ jako projekci P, což odpovídá proximálnímu operátoru pro indikátorovou funkci z tabulky 8.1.

$$\mathrm{prox}_{\tau g}(u) = P_{\left\{\mathbf{c}:|c_k - c_k^Q| \le \frac{\mathbf{Q}_k}{2},\, k=0,\ldots,63\right\}}(u). \tag{9.10}$$

Tato projekce způsobí, jak už bylo řečeno při popisu funkce g, že nové hodnoty c_k nebudou příliš vzdále od původních c_k^Q. Takže hodnoty nad limitem

$$c_k > c_k^Q + \frac{\mathbf{Q}_k}{2} \text{ přejdou na } c_k = c_k^Q + \frac{\mathbf{Q}_k}{2}$$

a hodnoty pod limitem

$$c_k < c_k^Q + \frac{\mathbf{Q}_k}{2} \text{ přejdou na } c_k = c_k^Q - \frac{\mathbf{Q}_k}{2}.$$

Proximální operátor pro funkci h_1 vyjádříme podle [12] jako

$$\text{prox}_{\sigma h_1^*}(u) = \text{prox}_{\sigma\|u\|_{1,2}^*}([u_h, u_v]) = \frac{[u_h, u_v]}{\max\left\{\sqrt{u_h^2 + u_v^2}, 1\right\}}, \tag{9.11}$$

Poznamenejme, že do argumentu operátoru $\text{prox}_{\sigma h_1^*}$ vstupuje jeden obraz u a jako výstup získáváme „dvojitý“ obraz o dvou maticích u_h a u_v. V algoritmu počítáme každou složku zvlášť, tj.

$$u_h^{(i,j)} = \frac{u_h^{(i,j)}}{\max\left\{\sqrt{\left(u_h^{(i,j)}\right)^2 + \left(u_v^{(i,j)}\right)^2}, 1\right\}}, \tag{9.12}$$

$$u_v^{(i,j)} = \frac{u_v^{(i,j)}}{\max\left\{\sqrt{\left(u_h^{(i,j)}\right)^2 + \left(u_v^{(i,j)}\right)^2}, 1\right\}}. \tag{9.13}$$

Pro vyjádření proximálního operátoru funkce h_2 potřebujeme Moreauovu identitu [12].

Tvrzení 9.1 (Moreauova identita). *Pro všechna $u \in \mathcal{U}_m$ a $\sigma \in \mathbb{R} > 0$ platí:*

$$prox_{\sigma h^*}(u) = u - \sigma prox_{h/\sigma}\left(\frac{u}{\sigma}\right), \tag{9.14}$$

kde h^* je konvexně sdružená funkce podle definice 8.7.

S použitím tvrzení 9.1 již můžeme psát

$$\text{prox}_{\sigma h_2^*}(u) = u - \sigma\text{prox}_{h_2/\sigma}\left(\frac{u}{\sigma}\right) = u - \sigma\text{prox}_{\iota_{\{\mathbf{x}:\mathbf{M}_x=\mathbf{M}_e\}}}\left(\frac{u}{\sigma}\right). \tag{9.15}$$

9.2.2. Volba parametrů

V algoritmu používáme čtyři parametry, které ovlivňují jak rychlost konvergence, tak i samotnou přesnost řešení. Jsou jimi σ, τ, ρ a ϵ.

Volba koeficientu σ proběhla experimentálně. Pro několik různých obrázků byly voleny hodnoty z intervalu $\sigma \in \langle 0,01; 0,50\rangle$. Nejrychleji algoritmus konvergoval k požadovanému řešení pro volbu $\sigma \in \langle 0,26; 0,30\rangle$.

Výraz pro výpočet hodnoty τ odvodíme pro náš případ $M = 2$ z podmínky v [12]:

$$\tau\sigma\left\|\sum_{m=1}^{M=2} L_m^* L_m\right\| \leq 1.$$

Jednoduchou úpravou vyjádříme τ jako

$$\tau \leq \frac{1}{\sigma\|L_1^* L_1 + L_2^* L_2\|}.$$

V části 9.1 jsme uvedli podobu funkcí L_1 a L_2, po dosazení dostáváme:

$$\begin{aligned}
\tau &\leq \frac{1}{\sigma\|\mathrm{IDCT}^* \circ \mathrm{D}^* \circ \mathrm{D} \circ \mathrm{IDCT} + \mathrm{IDCT}^*\mathrm{IDCT}\|} \\
\tau &\leq \frac{1}{\sigma\left(\|\mathrm{DCT} \circ \mathrm{D}^* \circ \mathrm{D} \circ \mathrm{IDCT}\| + \|\mathrm{ID}\|\right)} \\
\tau &\leq \frac{1}{\sigma\left(\|\mathrm{DCT}\| \cdot \|\mathrm{D}^* \circ \mathrm{D}\| \cdot \|\mathrm{IDCT}\| + 1\right)} \\
\tau &\leq \frac{1}{\sigma\left(1 \cdot 8 \cdot 1 + 1\right)} \\
\tau &\leq \frac{1}{9\sigma}
\end{aligned}$$

Parametr ρ volíme podle doporučení uvedeného v [12] z intervalu $(0, 2)$. Protože však neexistuje obecné doporučení pro konkrétní hodnotu, zvolili jsme ji experimentálně. Jako nejvýhodnější se ukázalo $\rho = 1,99$.

Pro parametr ϵ, kterým určujeme jak přesné řešení požadujeme a omezujeme tak i počet iterací algoritmu, jsme zvolili hodnotu $\epsilon = 1\mathrm{e}^{-4}$.

9.2.3. Ukončovací podmínka

Pro ukončení algoritmu volíme běžnou podmínka konvergence, tj. pokud pro zvolené $\epsilon \in \mathbb{R} > 0$ platí, že

$$\frac{\left\|x^n - x^{n-1}\right\|_2}{\left\|x^{n-1}\right\|_2} < \epsilon, \tag{9.16}$$

pak se algoritmus ukončí.

Převedeno do slov: pokud se normovaný rozdíl hodnot nového a předcházejícího řešení liší o méně než hodnotu ϵ, pak zřejmě s každou další iterací již nedochází ke zlepšení výsledku - vzhledem k požadované přesnosti ϵ - a můžeme algoritmus ukončit.

10. Aplikace v prostředí MATLAB

Vysvětleme nyní praktickou implementaci předchozí teorie do prostředí MATLAB. Algortimus samotný je napsán od základu, zatímco části jako převod obrazu do odstínů šedi, 2D DCT, Cannyho hranový detektor či metoda SSIM jsou implementovány s využitím již hotových funkcí MATLAB. Celý kód jak upraveného JPEG, tak i proximálního algoritmu zde uvádět nebudeme, ale jsou zahrnuty v příloze.

10.1. Kodér a dekodér

Algoritmus načítá jak černobílé, tak i barevné obrazy. Z důvodu zjednodušení však pracuje pouze s černobílými. Kódování pro barevné složky není implementováno a tedy i výstupy algoritmu jsou opět pouze v odstínech šedi. Na funkčnost ani názornost proximálního algoritmu, jehož funkčnost především testujeme, to však nemá vliv.

Také není zahrnuto kódování do binární reprezentace. Výstupem kodéru a vstupem dekodéru jsou kvantizované koeficienty. Opět to nemá vliv na funkčnost proximálního algoritmu, protože Huffmanovo kódování není ztrátové. Zjednodušujeme tak zároveň celý postup pouze na to nejpodstatnější.

10.1.1. Kodér

Upravený JPEG kodér pracuje takto:

1. **Nastavení parametrů:**
   - `quality` - nastavujeme požadovanou kvalitu komprese JPEG. Defaultně je nastaven na hodnotu 50, což odpovídá klasickému JPEG.
   - `nEdges` - udává procento hran v původním obrazu, které chceme zachovat. Defaultně je nastavn na hodnotu 0,5, což znamená 50% detekovaných hranových bodů.
   - `maxIterace` - nastavuje maximální počet iterací pro proximální algoritmus. To znamená, že algoritmus se ukončí po splnění ukončovací podmínky (viz odstavec 9.2.3) nebo dosažení tohoto maximálního počtu iterací.
2. **Načtení obrazu** ke zpracování.
3. **Kontrola velikosti obrazu.** Pokud je třeba, doplní se opakováním koncových pixelů na nejbližší násobek 8 v obou směrech.
4. **Převod do požadovaného barevného prostoru.** Místo jasové složky systému YC_BC_R zde využíváme převod na obraz v odstínech šedi prostřednictvím funkce `rgb2gray`. To proto, že nepracujeme s barevnými složkami C_B, C_R a tak je vhodné využít celé rozpětím 256 odstínů šedé, což, jak bylo uvedeno v teorii, není v systému YC_BC_R možné.
5. **Přetypování** z uint8 na double.
6. **Posun složek jasu** o -128.

7. **Detekce hran.** Pomocí funkce `edge` detekujeme hranové pixely v obraze a jejich zvoleného množství `nEdges` uložíme do samostatné matice. Poznamenejme, že hranovou matici jsme nejprve inicializovali s hodnotou -1000 v každém prvku. To proto, že jasovou složku jsme posunuli a hranový bod nyní může nabývat záporné hodnoty až -128.

8. **Načtení kvantizační matice** podle zvolené kvality `quality`.

9. **Kódování obrazu.** Pro jednotlivé bloky 8×8 pixelů provedeme klasické kódování JPEG (viz kapitola 5). V každé iteraci proběhne:

   - Načtení pixelů bloku.
   - Výpočet 2D DCT načteného bloku s využitím funkce pro transformační matici `dctmtx`.
   - Výpočet kvantizačních koeficientů.
   - Uložení bloku do matice přestavující „uložený“ obraz.

10.1.2. Dekodér

Upravený dekodér v podstatě sleduje kroky na zpět až na použití proximálního algoritmu. Postup je následující:

1. **Dekódování koeficientů.** Iterujeme pro jednotlivé bloky 8×8:

   - Načtení bloku z matice uložených kvantizační koeficientů.
   - Dekvantizace. Blok získaných transformačních koeficientů označme **B**.
   - Načtení odpovídajícího bloku matice s uloženými hodnotami hranových pixelů. Pro přehledností označme tento blok **E**.
   - Aplikace funkce s proximálním algoritmem (3). Nutnými vstupy jsou blok **B**, blok **E**, kvantizační matice a maximální počet iterací `maxIterace`. Pro celkové zrychlení algoritmu funkci pouštíme pouze pokud to má smysl, tedy pokud se v **E** najde alespoň jeden hranový bod. (To v programu prakticky znamená, že **E** bude mít alespoň jednu hodnotu větší než -1000).
   - Výpočet 2D IDCT hodnot získaných v předchozím kroku. Opět využíváme transformační matici `dctmtx`.
   - Přepsání odpovídajících hodnot v matici rekonstruovaného obrazu.

2. **Ořez na původní rozměr obrazu.**

3. **Posunutí jasu** zpět o +128.

4. **Převod zpět do číselného formátu uint8**, abychom byli schopní rekonstruovaný obraz zobrazit.

Následuje ještě výpočet hodnot MSE, PSNR a SSIM pro původní a rekonstruovaný obraz. Ty však nejsou součástí JPEG algoritmu a proto je do popisu postupu nezahrnujeme.

10.2. Proximální algoritmus

Programová implementace proximálního algortimu 3 v podstatě přesně kopíruje jeho teoretickou podobu a teoretické vyjádření proximálních operátorů. Navíc je přiložený skript podrobně popsán, takže čtenáře odkážeme přímo na něj.

Existuje pouze jeden rozdíl a to v ukončovací podmínce cyklu, kde byla použita proměnná `countIter`, která zajístí, že algoritmus se provede alespoň 100×. Během testování se totiž ukázalo, že vynucení alespoň nějakého menšího počtu iterací dokáže zlepšit celkovou vizuální kvalitu rekonstruovaného obrazu.

Uveďme ještě stručnou poznámku k použitému diferenčnímu operátoru D a jeho adjungovanému operátoru D^*. Způsob, jakým je tento operátor implementovaný ve skriptu, odpovídá rovnicím (9.4). Je však možné použít také jeho maticové vyjádření. Pro názornost uveďme jeho aplikaci v maticovém tvaru na vektor $\mathbf{x} = [x_1, x_2, x_3, x_4]$.

$$\mathrm{D}\mathbf{x}^T = \begin{bmatrix} -1 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} x_2 - x_1 \\ x_3 - x_2 \\ x_4 - x_3 \\ 0 \end{bmatrix}$$

Pro adjungovaný operátor D^* platí, že $\mathrm{D}^* = \mathrm{D}^T$. V maticovém vyjádření opět pro nějaké $n \in \mathbb{R}$

$$\mathrm{D}^* = \begin{bmatrix} -1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

Lze použít funkci MATLAB-u `diff`.

11. Hodnocení kvality obrazu

Před tím, než prakticky vyhodnotíme funkčnost a efektivitu našeho algoritmu je vhodné použitá kritéria nejdříve teoreticky uvést. Udělejme tak nyní.

Pro hodnocení kvality obrazu se využívají tzv. objektivní kritéria, která nezávisí na subjektivním vnímání. Mezi ně řadíme MSE a PSNR. Tyto metody hodnocení však silně korelují se subjektivním dojem. Byla proto vytvořena vylepšená kritéria jako je SSIM.

Dosud však neexistují dostatečně spolehlivé metody, které by byly schopné napodobit vnímání obrazu člověkem. Proto jediným dostatečně dobrým způsobem, jak hodnotit komprimační metody, zůstávají subjektivní testy.[56]

11.1. PSNR

PSNR (Peak signal-to-noise ratio), česky *špičkový poměr signálu k šumu*, je jednou z nejjednodušších a nejčastěji používaných metod pro hodnocení kvality obrazu. Obvykle se pro výpočet uvažuje pouze jasová složka, která má největší podíl na vizuální odlišnosti obrazů.[46]

Přestože dnes existují přesnější ukazatele kvality, je PSNR stále široce používán. Byla ukázáno, že PSNR odhaduje relativní percepční kvalitu poměrně dobře při použití fixního obsahu a kodeku napříč různými testovacími případy.[41]

Pro výpočet PSNR musíme nejprve spočítat střední kvadratickou chybu obrazu MSE (Mean Square Error). Pro obrazy $\mathbf{X}$ a $\mathbf{Y}$ o velikosti $M \times N$ pixelů je

$$\mathrm{MSE}(\mathbf{X}, \mathbf{Y}) = \frac{1}{MN} \sum_{m=0}^{M} \sum_{n=0}^{N} (\mathbf{X}(m,n) - \mathbf{Y}(m,n))^2. \tag{11.1}$$

Poznamenejme, že MSE se samo o sobě používá pro hodnocení kvality obrazu, přičemž platí, že čím menší hodnota, tím lepší je kvalita.

PSNR počítáme jako

$$\mathrm{PSNR}(\mathbf{X}, \mathbf{Y}) = 10 \log_{10} \left(\frac{(2^R - 1)^2}{\mathrm{MSE}(\mathbf{X}, \mathbf{Y})} \right), \tag{11.2}$$

kde R je počet bitů na pixel.

Označme Q hodnotu, pro kterou jsou dva obrazy od sebe téměř nerozeznatelné lidským zrakem. Za kvalitní obrazy považujeme takové, jejichž PSNR $\leq Q$. Při 8-bitové barevné hloubce ideální hodnota $Q \in [40, 50]$ decibelů.[60] Podle [20] je $Q = 60$dB pro 12-bitové a $Q = 80$dB pro 16-bitové obrazy. Hodnotu PSNR $= \infty$ dostáváme pro identické obrazy.

V prostředí MATLAB lze k výpočtu PSNR využít stejnojmennou funkci `psnr`.[47]

11.2. SSIM

Index strukturální podobnosti (SSIM, Structural Similarity Index) je metoda hodnocení kvality obrazu založená na porovnávání strukturálních vlastností.[66]

Východiskem je myšlenka, že jas povrchu pozorovaného objektu je výsledkem dopadu a odrazu světla, ale struktura objektů ve scéně je na tomto osvětlení nezávislá. Proto,

pokud chceme porovnávat strukturální vlastnosti obrazu, musíme vliv osvětlení oddělit. Strukturální informace v obraze definujeme jako vlastnosti, které reprezentují struktury nezávisle na jasu a kontrastu. Porovnává se jas l (luminance), kontrast c (contrast) a struktura s (structure). Výsledkem SSIM je hodnota v intervalu $[-1, 1]$, kde -1 znamená, že porovnávané obrazy jsou zcela odlišené a 1, že jsou totožné.

Nechť $\mathbf{X}$ a $\mathbf{Y}$ jsou dva obrazy, které byly vzájemně zarovnány. Jasovou složku l definujeme jako

$$l(\mathbf{X}, \mathbf{Y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \tag{11.3}$$

kde μ_x (resp. μ_y) je průměrná intenzita jasu

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{11.4}$$

Konstanta $C_1 = (K_1 L)^2$ je zahrnutá proto, abychom se vyhnuli nestabilitě v případě, kdy $\mu_x^2 + \mu_y^2$ je velmi blízko k nule. L je dynamický rozsah hodnot pixelů a $K_1 \ll 1$ je nějaká malá konstanta.

Kontrastní složku c definujeme jako

$$c(\mathbf{X}, \mathbf{Y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \tag{11.5}$$

kde σ_x (resp. σ_y) je standardní odchylka, kterou používáme jako odhad kontrastu signálu a je dána jako

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{1/2}. \tag{11.6}$$

Pro $K_2 \ll 1$ je konstanta $C_2 = (K_2 L)^2$.

Konečně, strukturální složku s definujeme výrazem

$$s(\mathbf{X}, \mathbf{Y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}, \tag{11.7}$$

kde σ_{xy} může být odhadnuté jako

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y). \tag{11.8}$$

SSIM pro obrazy $\mathbf{X}, \mathbf{Y}$ definujeme jako kombinaci rovnic (11.3), (11.5) a (11.7):

$$\mathrm{SSIM}(\mathbf{X}, \mathbf{Y}) = [l(\mathbf{X}, \mathbf{Y})^\alpha \cdot c(\mathbf{X}, \mathbf{Y})^\beta \cdot s(\mathbf{X}, \mathbf{Y})^\gamma], \tag{11.9}$$

kde $\alpha > 0, \beta > 0$ a $\gamma > 0$ jsou parametry použité k úpravě relativní důležitosti jednotlivých komponent.[66]

V programu MATLAB je pro výpočet indexu SSIM k dispozici funkce `ssim`.[49]

12. Výsledky testování algoritmu

Aplikaci algoritmu nyní názorně ukážeme na třech srovnávacích obrázcích. Pro každý z nich byl algoritmus pouštěn pro zvolené množství zachovaných hranových pixelů. 25%, 50%, 75% a 100% detekovaných bodů. Kvalita Q udávaná pro JPEG zůstávala nezměněná na $Q = 50$. Byl měřen čas trvání algoritmu, hodnota MSE, PSNR a SSIM. Zaznamenané hodnoty jsou porovnány vzhledem k hodnotám naměřených pro originální algoritmus.

Algoritmus byl následně puštěn také na sadě standardizovaných přirozených obrazů stažených z [73] a doplněných o několik snímků z [71]. Obrazy jsou k nalezení v souborech přiložených k práci.

Na obrázku 12.1 je velmi pěkně vidět, výrazné zlepšení vizuální kvality po použití upraveného algoritmu. Blokové artefakty, až na několik málo míst, se již v obrázku nenachází. Vysvětlení toho, proč přesto někde artefakty zůstaly, spočívá v náhodném výběru hranových bodů. Je totiž vidět, že bloky zůstávají zejména v oblastech, kde kodér zachoval bodů méně. Dekodér následně krok proximálního algoritmu přeskočil, protože nebyly nalezeny žádné referenční hrany. Blok tedy zůstal beze změny tak, jak jej dekódoval neupravený JPEG.

Případ obrázku 12.2 je poměrně odlišný od toho předchozího. Originální obrázek obsahuje mnohem více šumu a není tak hladký jako ten první. Hrany nejsou tak viditelné a chyby rekonstrukce se vyskytují v rámci celého obrázku. I zde byl algoritmus úspěšný. Bohužel je zřejmé, že cenou se stalo celkové snížení ostrosti obrázku díky jeho vyhlazení.

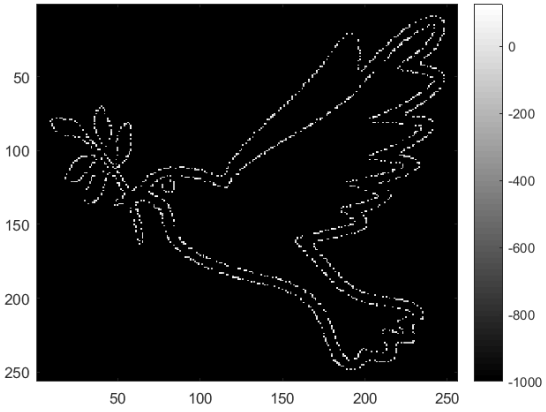
Podobně jako v prvním případě, obrázky 12.3 ukazují, že implementovaný proximální algoritmus dokázal, až na výjimky, chyby v obraze odstranit.

Tabulka 12.1 srovnává hodnoty testovacích metod pro 3 předchozí případy. Hodnoty naměřené pro klasický algoritmu jsou označené jako „1“, hodnoty obrázků dekódovaných upraveným algoritmem jako „2“.

	Dove	**Statue**	**Text**
Čas 1	0,017	0,019	0,031
Čas 2	78,166	119,142	73,991
MSE1	72,457	19,664	38,575
MSE2	9,210	19,206	19,662
PSNR1	29,530	35,194	32,268
PSNR2	38,489	35,297	35,195
SSIM1	0,826	0,544	0,803
SSIM2	0,896	0,547	0,845

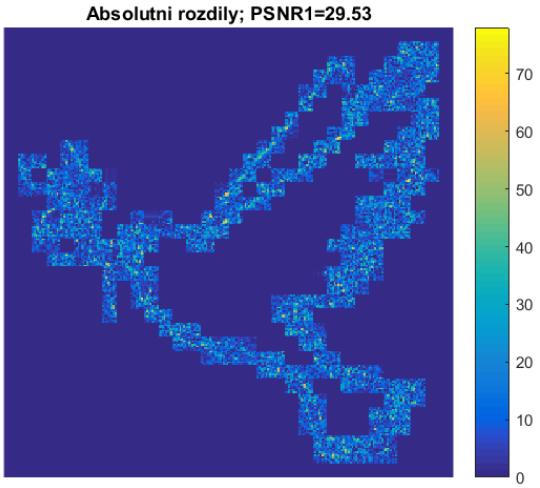
Tabulka 12.1: Hodnoty MSE, PSNR a SSIM pro porovnávané algoritmy při zachování 50% hranových bodů.

(a) Originální obrázek.



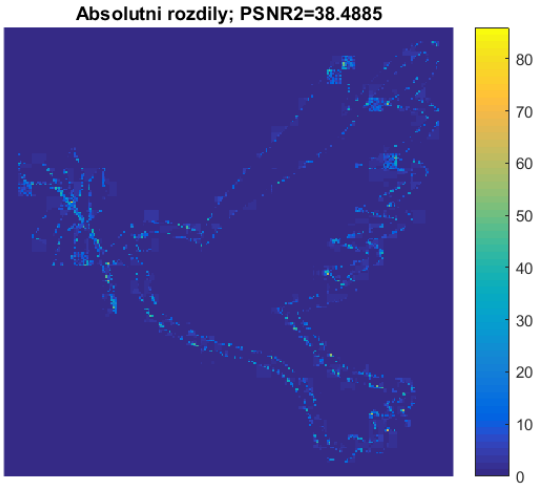
(b) Vybrané detekované hrany.

(c) Výsledek po použití klasického JPEG algoritmu.



(d) Rozdíl původním obrázkem a výsledkem klasického algoritmu.

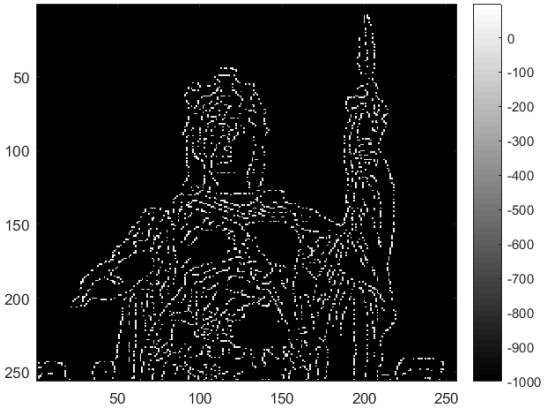
(e) Výsledek po použití upraveného JPEG algoritmu.



(f) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

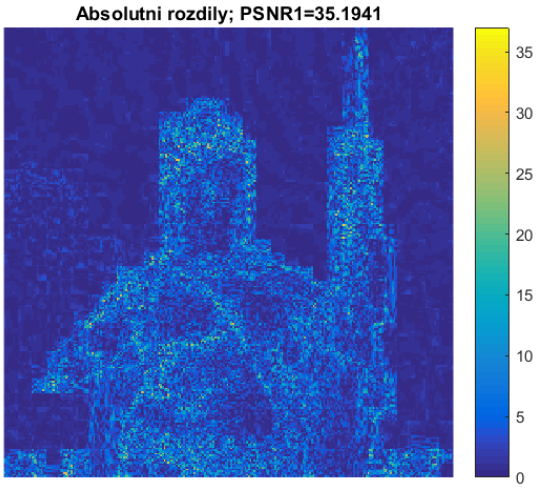
Obrázek 12.1: Srovnání obrázku „Holubice“ po rekonstrukci při zachování 50% hranových pixelů.

(a) Originální obrázek.



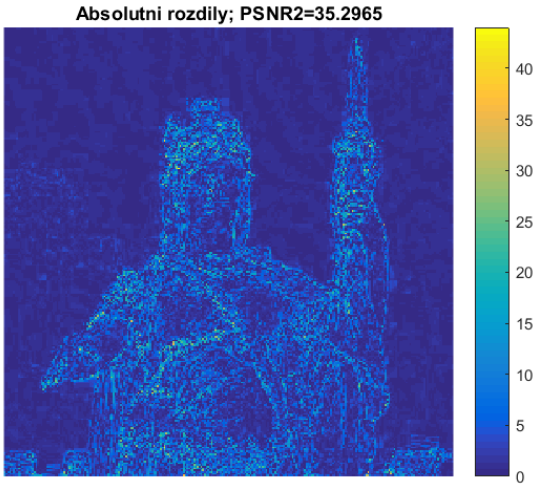
(b) Vybrané detekované hrany.

(c) Výsledek po použití klasického JPEG algoritmu.



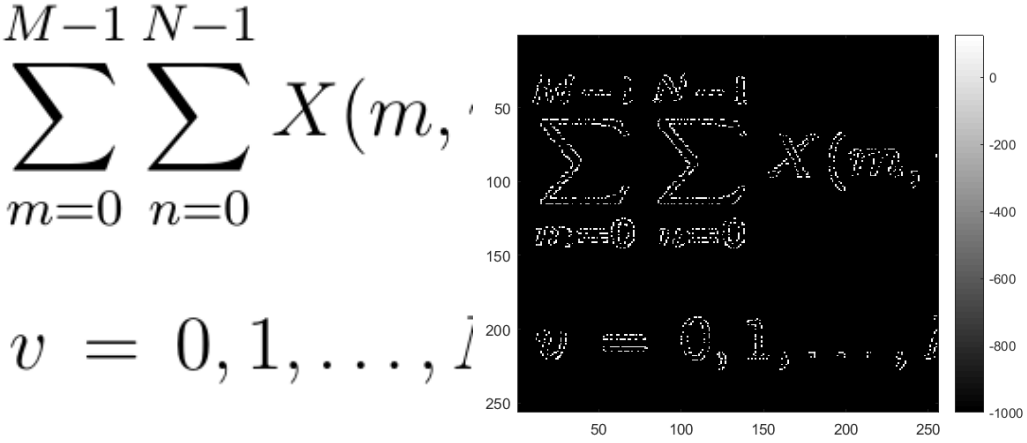
(d) Rozdíl původním obrázkem a výsledkem klasického algoritmu.

(e) Výsledek po použití upraveného JPEG algoritmu.



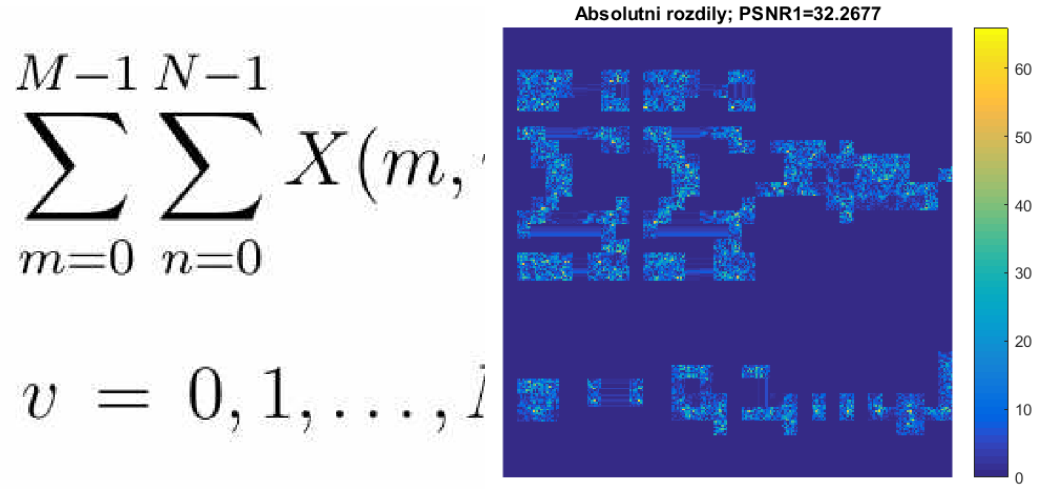
(f) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

Obrázek 12.2: Srovnání obrázku „Socha“ po rekonstrukci při zachování 50% hranových pixelů.



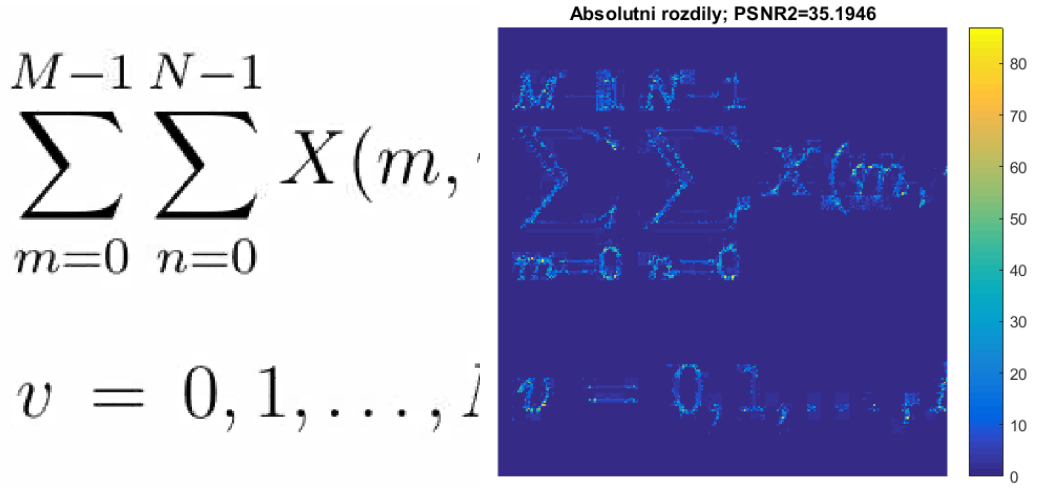
(a) Originální obrázek.

(b) Vybrané detekované hrany.



(c) Výsledek po použití klasického JPEG algoritmu.

(d) Rozdíl původním obrázkem a výsledkem klasického algoritmu.



(e) Výsledek po použití upraveného JPEG algoritmu.

(f) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

Obrázek 12.3: Srovnání obrázku „Text“ po rekonstrukci při zachování 50% hranových pixelů.

Z těchto tří obrázků a z uvedených tabulek s hodnotami testovacích metod je zřejmé, že pokud jde o obrazy, které nemají větší barevné plochy bez výrazných změn, algoritmus má tendenci obrazy spíše rozmazat. Je sice schopen odstranit artefakty na hranách tak, jak je požadováno, ale za cenu redukce poměrně velkého množství detailů v obraze. Naproti tomu, zlepšení pro obrazy typu „Holubice“ je znatelné.

Tomu odpovídají i výsledky z testování na skupině 43 standardizovaných obrazů. Rozdíly ve sledovaných metrikách MSE, PSNR a SSIM jsou oproti původnímu JPEG algoritmu jen velmi malé. V následující tabulkách 12.2 a 12.3 jsou průměrné hodnoty měření. Opět platí stejné označení jedničkou pro hodnoty původního algoritmu a dvojkou pro nový algoritmus.

	Zachováno 25% bodů hran			Zachováno 50% bodů hran		
	MSE	**PSNR**	**SSIM**	**MSE**	**PSNR**	**SSIM**
Průměry 1	29,447	34,525	0,699	29,447	34,525	0,699
Průměry 2	33,406	33,948	0,675	30,875	34,310	0,681
Mediány 1	23,599	34,402	0,722	23,599	34,402	0,722
Mediány 2	26,972	33,822	0,694	24,143	34,303	0,697

Tabulka 12.2: Průměrné hodnoty MSE, PSNR a SSIM pro testovací sadu obrázků.

	Zachováno 75% bodů hran			Zachováno 100% bodů hran		
	MSE	**PSNR**	**SSIM**	**MSE**	**PSNR**	**SSIM**
Průměry 1	29,447	34,525	0,699	29,447	34,525	0,699
Průměry 2	28,204	34,724	0,689	25,560	35,161	0,697
Mediány 1	23,599	34,402	0,722	23,599	34,402	0,722
Mediány 2	22,354	34,637	0,703	20,618	34,988	0,709

Tabulka 12.3: Průměrné hodnoty MSE, PSNR a SSIM pro testovací sadu obrázků.

V tabulce 12.4 jsou délky trvání algoritmů. Označení se řídí opět stejnou logikou. Jde o průměrné hodnoty v závislosti na velikosti testovacích obrázků.

	rozlišení	**25%**	**50%**	**75%**	**100%**
v1	256x256	0,006	0,010	0,006	0,007
v2	pixelů	58,602	64,506	67,885	72,004
v1	512x512	0,023	0,035	0,024	0,024
v2	pixelů	305,862	358,188	348,384	370,122
v1	1024x1024	0,096	0,109	0,095	0,093
v2	pixelů	1133,492	1308,740	1329,908	1422,879

Tabulka 12.4: Hodnoty MSE, PSNR a SSIM pro porovnávané algoritmy při zachování 50% hranových bodů.

Dalších z významných nedostatků algoritmu je jeho podstatně menší rychlost oproti klasickému algoritmu JPEG. Ta se roste s množstvím detekovaných hran a s velikostí obrazu, což odpovídá množství bloků, pro které se algoritmus provádí.

Otázkou, kterou jsme se v této práci nezabývali, je také způsob uchování hodnot vybraných hranových pixelů. Nyní jsme je pouze načetli do matice a ihned použili při dekódování. Pokud bychom ale předpokládali, že vylepšený algoritmus bude sloužit k zobrazování obrazů na různých zařízeních, je nutné tyto hodnoty nějakým vhodným způsobem také

uložit do generovaného souboru. Tím však také rostou nároky na paměť, protože tyto hodnoty nijak nekomprimujeme. Bylo tedy potřeba najít vhodný způsob jejich kódování do binární reprezentace a vyšetřit vliv na konečnou velikost souboru komprimovaného obrazu.

Pokud totiž odstraníme nechtěné blokové artefakty, ale zároveň podstatně zvýšíme velikost souboru, postrádá prakticky použití JPEG komprese smysl. Bylo by tak mnohem lepší využít některý z bezeztrátových formátů.

Nicméně, jak už bylo řečeno, pro specifické typy obrazů (např. snímky text, či jednoduché vektorové grafiky) je algoritmus efektivní a pracuje podle stanovených požadavků. Jeho využití v této oblasti lze dále zkoumat.

Závěr

Cílem této diplomové práce bylo seznámit se detailně s principem a praktickým fungováním algoritmu JPEG, pochopit proč při rekonstrukci obrazu vznikají na hranách blokové artefakty a za využití znalostí o řídkých reprezentacích signálu, proximálním operátoru a proximálních algoritmech upravit originální JPEG kodér a dekodér tak, aby se tyto artefakty v rekonstruovaném obraze neobjevovaly.

Práce se nejprve věnovala definici problému a navržení způsobu jeho řešení. Následně byl popsán obecný mechanismus blokového transformačního kódování včetně popisu používaných barevných prostorů. Dále byla podrobně popsána diskrétní kosinová transformace, která je klíčovým prvkem kompresních algoritmů ve zpracování signálů. Byl podrobně popsán standard JPEG, včetně jeho praktického fungování. Jedna z kapitol byla věnována také detekci hran za využití Cannyho algoritmu.

Druhá část práce se věnovala řídkým reprezentacím signálu a z nich vycházejícímu proximálnímu operátoru. Byla vysvětlena jeho aplikace v proximálních algoritmech a byl popsán princip těchto algoritmů.

Nakonec byl navržen a sestaven proximální algoritmus, řešící optimalizační úlohu vedoucí k odstranění nechtěných blokových artefaktů po dekódování v obraze. Poté byly prezentovány výsledky praktického použití navrženého algoritmu.

Upravený JPEG algoritmus svoji úlohu splnil. I když, jak bylo názorně ukázáno v sekci s výsledky, jeho úspěšnost z velké části závisí na volbě specifických případů. Při použití na obecný přirozený obraz objektivní ukazatele MSE, PSNR a SSIM neukázaly výrazné zlepšení kvality obrazu, jaké bylo očekáváno.

Literatura

[1] .PGF File Extension: Progressive Graphics File. *FileInfo.com* [online]. Updated March 2, 2018 [cit. 2023-05-26]. Dostupné z: https://fileinfo.com/extension/pgf

[2] .WEBP File Extension: WebP Image. *FileInfo.com* [online]. Updated April 21, 2022 [cit. 2023-05-26]. Dostupné z: https://fileinfo.com/extension/webp

[3] ADOBE. JPEG files. *Adobe.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.adobe.com/creativecloud/file-types/image/raster/jpeg-file.html

[4] ADOBE. JPEG vs. JPEG 2000. *Adobe.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.adobe.com/creativecloud/file-types/image/comparison/jpeg-vs-jpeg-2000.html

[5] AHMED, N., T. NATARAJAN a K.R. RAO. Discrete Cosine Transform. *IEEE Transactions on Computers* [online]. 1974, C-23(1), 90-93 [cit. 2023-05-26]. ISSN 0018-9340. Dostupné z: doi:10.1109/T-C.1974.223784

[6] AHMED, Nasir. How I came up with the discrete cosine transform. *Digital Signal Processing* [online]. 1991, 1(1), 4-5 [cit. 2023-05-26]. ISSN 10512004. Dostupné z: doi:10.1016/1051-2004(91)90086-Z

[7] BRAGANZA, Sherman a Miriam LEESER. The 1D Discrete Cosine Transform For Large Point Sizes Implemented On Reconfigurable Hardware. In: *2007 IEEE International Conf. on Application-specific Systems, Architectures and Processors (ASAP)* [online]. IEEE, 2007, 2007, s. 101-106 [cit. 2023-05-26]. ISBN 978-1-4244-1026-2. Dostupné z: doi:10.1109/ASAP.2007.4429965

[8] BT.601. Studio encoding parameters of digital television for standard 4:3 and wide screen 16:9 aspect ratios. 4. ITU-R., 2011. Dostupné také z: https://www.itu.int/rec/R-REC-BT.601

[9] BULL, David R. a Fan ZHANG. Lossless compression methods. In: *Intelligent Image and Video Compression* [online]. Elsevier, 2021, 2021, s. 225-270 [cit. 2023-05-26]. ISBN 9780128203538. Dostupné z: doi:10.1016/B978-0-12-820353-8.00016-5

[10] CANNY, John. A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 1986, PAMI-8(6), 679-698 [cit. 2023-05-26]. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.1986.4767851

[11] *Canny Edge Detection: 09gr820* [online]. 23 March 2009 [cit. 2023-05-26]. Dostupné z: http://kiwi.bridgeport.edu/cpeg585/CannyEdgeDetector.pdf

[12] CONDAT, Lauren. A Generic Proximal Algorithm for Convex Optimization—Application to Total Variation Minimization. *IEEE Signal Processing Letters* [online]. 2014, 21(8), 985-989 [cit. 2023-05-26]. ISSN 1070-9908. Dostupné z: doi:10.1109/LSP.2014.2322123

[13] DUVAL, Laurent. JPEG DCT padding. In: *dsp.stackexchange.com* [online]. 6 Nov 2016 [cit. 2023-05-26]. Dostupné z: https://dsp.stackexchange.com/questions/35339/jpeg-dct-padding/35343#35343

[14] FORD, Adrian a Alan ROBERTS. *Colour Space Conversions* [online]. 11 August 1998 [cit. 2023-05-26]. Dostupné z: https://opticsoflife.org/pdfs/tech/colorconversion.pdf

[15] FSV CVUT. *Ztrátové formáty* [online]. [cit. 2023-05-26]. Dostupné z: https://gis.fsv.cvut.cz/kartografie/4-4-3-ztratove-formaty.php

[16] GONZALEZ, Rafael C. a Richard E. WOODS. *Digital image processing*. 3rd ed. Upper Saddle River, N.J.: Prentice Hall, c2008. ISBN 978-0-13-168728-8. Dostupné z: https://dl.ebooksworld.ir/motoman/Digital.Image.Processing.3rd.Edition.www.EBooksWorld.ir.pdf

[17] HRBÁČEK, Radek, Pavel RAJMIC, Vítězslav VESELÝ a Jan ŠPIŘÍK. Řídké reprezentace signálů: úvod do problematiky. *Elektro revue* [online]. 2011, 13. 9. 2011, 13(5) [cit. 2023-05-26]. ISSN 1213-1539. Dostupné z: http://www.elektrorevue.cz/cz/download/ridke-reprezentace-signalu--uvod-do-problematiky/.e.cz/

[18] HSIA, Shih-Chang, Szu-Hong WANG a Chia-Jung CHEN. Fast search real-time face recognition based on DCT coefficients distribution. *IET Image Processing* [online]. 2020, 14(3), 570-575 [cit. 2023-05-26]. ISSN 1751-9667. Dostupné z: doi:10.1049/iet-ipr.2018.6175

[19] CHAMBOLLE, Antonin a Thomas POCK. A First-Order Primal-Dual Algorithm for Convex Problems with Applications to Imaging. Journal of Mathematical Imaging and Vision [online]. 2011, 40(1), 120-145 [cit. 2023-05-26]. ISSN 0924-9907. Dostupné z: doi:10.1007/s10851-010-0251-1

[20] CHERVYAKOV, Nikolay, Pavel LYAKHOV a Nikolay NAGORNOV. Analysis of the Quantization Noise in Discrete Wavelet Transform Filters for 3D Medical Imaging. *Applied Sciences* [online]. 2020, 10(4) [cit. 2023-05-26]. ISSN 2076-3417. Dostupné z: doi:10.3390/app10041223

[21] CHIERCHIA, G, E CHOUZENOUX, P.L. COMBETTES a J.-C. PESQUET. *The Proximity Operator Repository* [online]. 2016 [cit. 2023-05-25]. Dostupné z: http://proximity-operator.net/index.html

[22] IBRAHEEM, Noor A., Mokhtar M. HASAN, Rafiqul Z. KHAN a Pramod K. MISHRA. Understanding Color Models: A Review. *ARPN Journal of Science and Technology* [online]. 2012, April 2012, 2(3), 265-275 [cit. 2023-05-26]. ISSN 2225-7217. Dostupné z: https://www.researchgate.net/publication/266462481_Understanding_Color_Models_A_Review

[23] INSTITUT BIOSTATISTIKY A ANALÝZ LÉKAŘSKÉ FAKULTY MASARYKOVY UNIVERZITY. 3 Diskrétní Fourierova transformace (DFT). E-learningová učebnice [online]. [cit. 2023-05-26]. Dostupné z: https://portal.matematickabiologie.cz/index.php?pg=analyza-a-modelovani-dynamickych-biologickych-dat--signaly-a-linearni-systemy--casov

[24] ISO/IEC 10918-1:1994/COR 1:2005. *Information technology — Digital compression and coding of continuous-tone still images: Requirements and guidelines — Technical Corrigendum 1: Patent information update.* 1. 2005.

[25] ISO/IEC 15444-1:2000. *Information technology — JPEG 2000 image coding system — Part 1: Core coding system.* 1. 2000.

[26] ISO/IEC 29199-2:2009. *JPEG XR image coding system — Part 2: Image coding specification.* 1. 2009.

[27] ISO/IEC 18477-1:2015. *Information technology — Scalable compression and coding of continuous-tone still images — Part 1: Scalable compression and coding of continuous--tone still images.* 1. 2015.

[28] ISO/IEC 21122-1:2019. *Information technology — JPEG XS low-latency lightweight image coding system — Part 1: Core coding system.* 1. 2019.

[29] ISO/IEC 15444-15:2019. *Information technology — JPEG 2000 image coding system — Part 15: High-Throughput JPEG 2000.* 1. 2019.

[30] ISO/IEC 18181-2:2021. *Information technology — JPEG XL image coding system — Part 2: File format.* 1. 2021.

[31] ISO/IEC 18181-1:2022. *Information technology — JPEG XL image coding system — Part 1: Core coding system.* 1. 2022.

[32] ISO/IEC 15444-17. *Information technology — JPEG 2000 image coding system — Part 17: Extensions for coding of discontinuous media.* 1.

[33] JACK, Keith. *Video demystified: a handbook for the digital engineer.* 5th ed. Boston: Newnes, c2007. ISBN 978-0-7506-8395-1.

[34] JAIN, Anil K. A Sinusoidal Family of Unitary Transforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence* [online]. 1979, PAMI-1(4), 356-365 [cit. 2023-05-26]. ISSN 0162-8828. Dostupné z: doi:10.1109/TPAMI.1979.4766944

[35] JAIN, Ramesh a Rangachar KASTURI. *Machine vision.* Boston: McGraw-Hill, 1995. ISBN 0-07-032018-7. Dostupné z: https://cse.usf.edu/~r1k/MachineVisionBook/MachineVision.htm

[36] JOOHEUNG LEE, N. VIJAYKRISHNAN a M.J. IRWIN. Inverse discrete cosine transform architecture exploiting sparseness and symmetry properties. *IEEE Transactions on Circuits and Systems for Video Technology* [online]. 2006, 16(5), 655-662 [cit. 2023-05-26]. ISSN 1051-8215. Dostupné z: doi:10.1109/TCSVT.2006.873155

[37] *JPEG* [online]. [cit. 2023-05-26]. Dostupné z: https://jpeg.org/

[38] KHAYAM, Syed Ali. The Discrete Cosine Transform (DCT): Theory and Application1 [online]. Course Notes, Department of Electrical & Computer Engineering, 2003 [cit. 2023-05-26]. Dostupné z: https://www.researchgate.net/publication/242275100_The_Discrete_Cosine_Transform_DCT_Theory_and_Application1

[39] KHOSLA, Dishant. Design of Hybrid Compression Model using DWT-DCT-HUFFMAN Algorithms for Compression of Bit Stream. International Journal of Engineering and Technical Research [online]. 2012, 1. 1-7. [cit. 2023-05-26].

[40] KLÍMA, Miloš, Martin BERNAS, Jiří HOZMAN a Pavel DVOŘÁK. Zpracování obrazové informace. Praha: České vysoké učení technické, 1996. ISBN 80-01-01436-3.

[41] KORHONEN, Jari a Junyong YOU. Peak signal-to-noise ratio revisited: Is simple beautiful?. In: 2012 *Fourth International Workshop on Quality of Multimedia Experience* [online]. IEEE, 2012, 2012, s. 37-38 [cit. 2023-05-26]. ISBN 978-1-4673-0726-0. Dostupné z: doi:10.1109/QoMEX.2012.6263880

[42] LANE, Tom. INDEPENDENT JPEG GROUP. *JPEG image compression FAQ, part 1/2* [online]. Aktualizováno 28.3.1999 [cit. 2023-05-26]. Dostupné z: http://www.faqs.org/faqs/jpeg-faq/part1/index.html

[43] MANNAN, Sam. *Discrete Cosine Transform* [online]. Elsevier, 1990 [cit. 2023-05-26]. ISBN 978-0-08-092534-9. Dostupné z: doi:10.1016/C2009-0-22279-3

[44] Discrete Cosine Transform. THE MATHWORKS, INC. *Mathworks.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.mathworks.com/help/images/discrete-cosine-transform.html

[45] Edge Detection. THE MATHWORKS, INC. *Mathworks.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.mathworks.com/help/images/edge-detection.html

[46] PSNR: Compute peak signal-to-noise ratio (PSNR) between images. THE MATHWORKS, INC. *Mathworks.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.mathworks.com/help/vision/ref/psnr.html

[47] Psnr: Peak signal-to-noise ratio (PSNR). THE MATHWORKS, INC. *Mathworks.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.mathworks.com/help/images/ref/psnr.html

[48] Rgb2ycbcr: Convert RGB color values to YCbCr color space. THE MATHWORKS, INC. *Mathworks.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.mathworks.com/help/images/ref/rgb2ycbcr.html#bvidgw5-YCBCR

[49] Ssim: Structural similarity (SSIM) index for measuring image quality. THE MATHWORKS, INC. *Mathworks.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.mathworks.com/help/images/ref/ssim.html

[50] MOREAU, J.J.: Fonctions convexes duales et points proximaux dans un espace hilbertien. *Comptes rendus hebdomadaires des se´ances de l'Acade´mie des sciences.* 1962, 255, 2897–2899 [cit. 2023-05-25].

[51] NGUYEN, Cung. *Reconstruct Huffman Code Table From the DTH segment.* 2001. Dostupné také z: https://www.ece.ucdavis.edu/cerl/wp-content/uploads/sites/14/2015/09/GenHuffCodes.pdf. Progress Report. University of California-Davis. Vedoucí práce G. Robert Redinbo, Ph.D.

[52] PAL, N.R. a S.K. PAL. Entropy: a new definition and its applications. *IEEE Transactions on Systems, Man, and Cybernetics* [online]. 21(5), 1260-1270 [cit. 2023-05-26]. ISSN 00189472. Dostupné z: doi:10.1109/21.120079

[53] PESQUET, J.-C. a Patrick L. COMBETTES. *Proximal Splitting Methods in Signal Processing.* 2009. Dostupné z: doi:10.48550/arXiv.0912.3522

[54] PLATANIOTIS, Konstantinos N. a Anastasios N. VENETSANOPOULOS. *Color Image Processing and Applications* [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000 [cit. 2023-05-26]. Digital Signal Processing. ISBN 978-3-642-08626-7. Dostupné z: doi:10.1007/978-3-662-04186-4

[55] POYNTON, Charles. *Chroma subsampling notation* [online]. San Francisco, 2003, 3 [cit. 2023-05-26]. Dostupné z: https://poynton.ca/PDFs/Chroma_subsampling_notation.pdf

[56] RAJMIC, Pavel a Jiří SCHIMMEL. *Moderní počítačová grafika.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2013. ISBN 978-80-214-4906-0.

[57] RAJMIC, P. *Řídké a nízkohodnostní reprezentace signálů s aplikacemi.* Brno, 2014. Habilitační práce. VUT v Brně.

[58] RÉMY, Abergel. *Legendre-Fenchel and Rockafellar duality* [online]. [cit. 2023-05-26]. Dostupné z: https://helios2.mi.parisdescartes.fr/~rabergel/legendre-fenchel-rockafellar-duality-4tv.html

[59] RICHTER, Thomas, Alessandro ARTUSI a Touradj EBRAHIMI. JPEG XT: A New Family of JPEG Backward-Compatible Standards. *IEEE MultiMedia* [online]. 2016, 23(3), 80-88 [cit. 2023-05-26]. ISSN 1070-986X. Dostupné z: doi:10.1109/MMUL.2016.49

[60] SHEIKH, H.R. a A.C. BOVIK. Image information and visual quality. *IEEE Transactions on Image Processing* [online]. 2006, 15(2), 430-444 [cit. 2023-05-26]. ISSN 1057-7149. Dostupné z: doi:10.1109/TIP.2005.859378

[61] T.871. Information technology – Digital compression and coding of continuous-tone still images: JPEG File Interchange Format (JFIF). ITU-T., 2012. Dostupné z: https://www.itu.int/rec/T-REC-T.871-201105-I/en

[62] TAN, Guanghua, Sanyuan ZHANG a Yin ZHANG. Shape Morphing for Point Set Surface Based on Vertex Deformation Gradient. In: *2009 WRI World Congress on Software Engineering* [online]. IEEE, 2009, 2009, s. 466-471 [cit. 2023-05-26]. ISBN 978-0-7695-3570-8. Dostupné z: doi:10.1109/WCSE.2009.109

[63] TRIANA, Juan a Luis FERRO. Finite difference methods in image processing. *Selecciones Matemáticas* [online]. 2021, 8(02), 411-416 [cit. 2023-05-26]. ISSN 24111783. Dostupné z: doi:10.17268/sel.mat.2021.02.17

[64] TROPP, Joel A a Stephen J WRIGHT. Computational Methods for Sparse Solution of Linear Inverse Problems. *Proceedings of the IEEE* [online]. 2010, 98(6), 948-958 [cit. 2023-05-25]. ISSN 0018-9219. Dostupné z: doi:10.1109/JPROC.2010.2044010

[65] VASUKI, A. a P.T. VANATHI. A review of vector quantization techniques. *IEEE Potentials* [online]. 2006, 25(4), 39-47 [cit. 2023-05-25]. ISSN 0278-6648. Dostupné z: doi:10.1109/MP.2006.1664069

[66] WANG, Z., A.C. BOVIK, H.R. SHEIKH a E.P. SIMONCELLI. Image Quality Assessment: From Error Visibility to Structural Similarity. *IEEE Transactions on Image Processing* [online]. 2004, 13(4), 600-612 [cit. 2023-05-26]. ISSN 1057-7149. Dostupné z: doi:10.1109/TIP.2003.819861

[67] WU, Xiang, Weibo YU, Xiaotong LIU a Keping LIU. A Newly Improved Canny Algorithm of Image Edge Detection. In: *Proceedings of the 6th International Conference on Information Engineering for Mechanics and Materials* [online]. Paris, France: Atlantis Press, 2016, 2016, - [cit. 2023-05-26]. ISBN 978-94-6252-244-2. Dostupné z: doi:10.2991/icimm-16.2016.68

[68] YAN, Eddie Q., ZHANG, Kaiyuan, WANG, Xi, STRAUSS, Karin a Luis Ceze. Customizing Progressive JPEG for Efficient Image Storage. *USENIX Workshop on Hot Topics in Storage and File Systems*, 2017 [cit. 2023-05-26]. Dostupné z: https://www.usenix.org/system/files/conference/hotstorage17/hotstorage17-paper-yan.pdf

[69] YUAN, Pei, Zhong-hua HU, Gan-xin OUYANG a Xue-jun ZHANG. Color Space Conversion from RGB to YCbCr based on FPGA. *DEStech Transactions on Computer Science and Engineering* [online]. 2018, (pcmm) [cit. 2023-05-26]. ISSN 2475-8841. Dostupné z: doi:10.12783/dtcse/pcmm2018/23660

[70] Dove emoji clipart. In: *Creazilla.com* [online]. [cit. 2023-05-26]. Dostupné z: https://creazilla.com/nodes/45800-dove-emoji-clipart. (CC BY 4.0) Upraveno.

[71] GONZALEZ, WOODS a EDDINS. Image Databases. *ImageProcessingPlace.com* [online]. [cit. 2023-05-26]. Dostupné z: https://www.imageprocessingplace.com/root_files_V3/image_databases.htm

[72] RAJMIC, Pavel. FEKT VUT V BRNĚ. *Matlab-grafika: github repository* [online]. 2019 [cit. 2023-05-26]. Dostupné z: https://github.com/rajmic/matlab-grafika

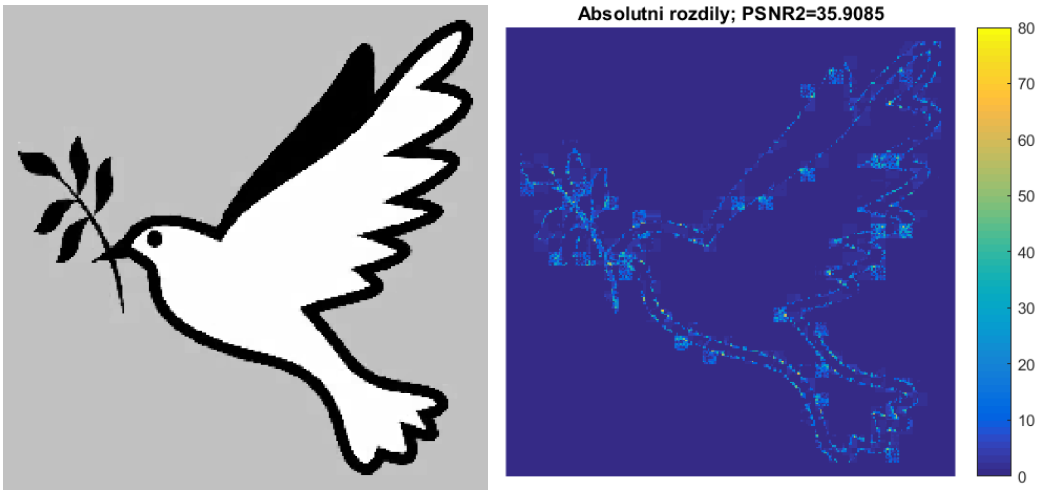
[73] The USC-SIPI Image Database. *USC Viterbi* [online]. University of Southern California, 1981 [cit. 2023-05-26]. Dostupné z: https://sipi.usc.edu/database/

SEZNAM ZKRATEK

Zkratka	Slovní popis
DCT	discrete cosine transform - diskrétní kosinová transformace
DFT	discrete Fourier transform - diskrétní fourierova transformace
FFT	fast Fourier transform - rychlá Fourieriova transformace
IDCT	inverse discrete cosine transform - inverzní diskrétní kosinová transformace
RGB	barevný prostor red-grean-blue
WHT	Walsh-Hadamard transform - Walsh-Hadamardova transformace
YC_BC_R	barevný prostor YCBCR
YUV	barevný prosotor YUV

PŘÍLOHY

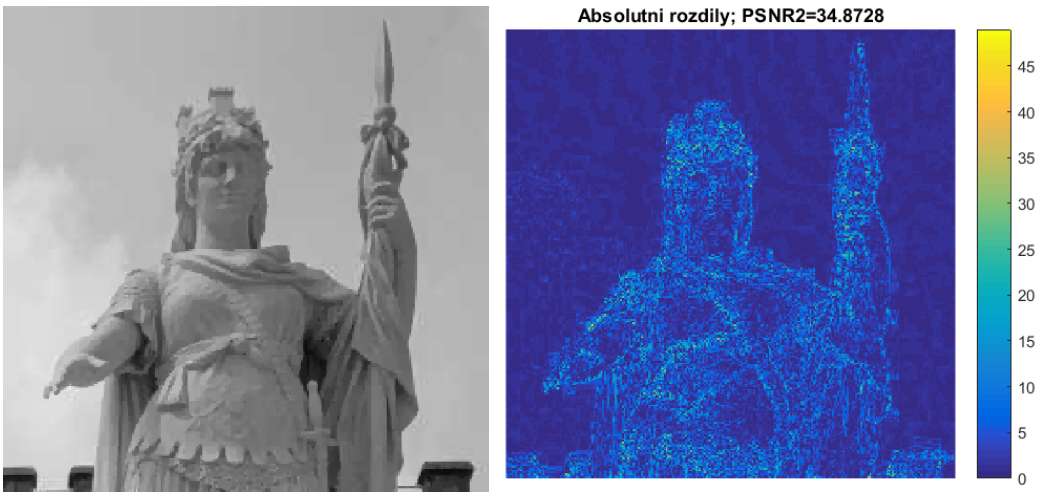
Výsledné obrázky pro různé nastavení počtu zachovaných hraničních bodů



(a) Výsledek po použití upraveného JPEG algoritmu.

(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

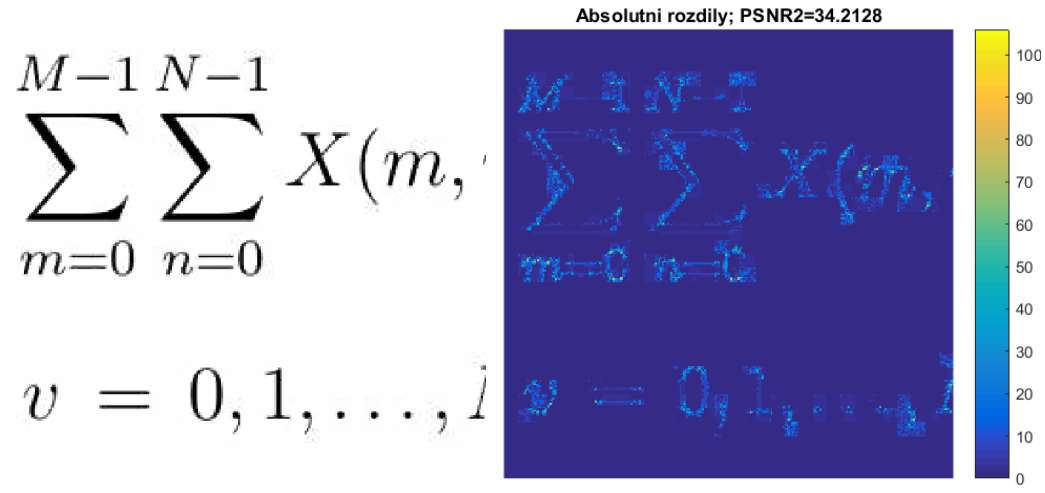
Obrázek 12.4: Srovnání obrázků po rekonstrukci při zachování 25% hranových pixelů.



(a) Výsledek po použití upraveného JPEG algoritmu.

(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

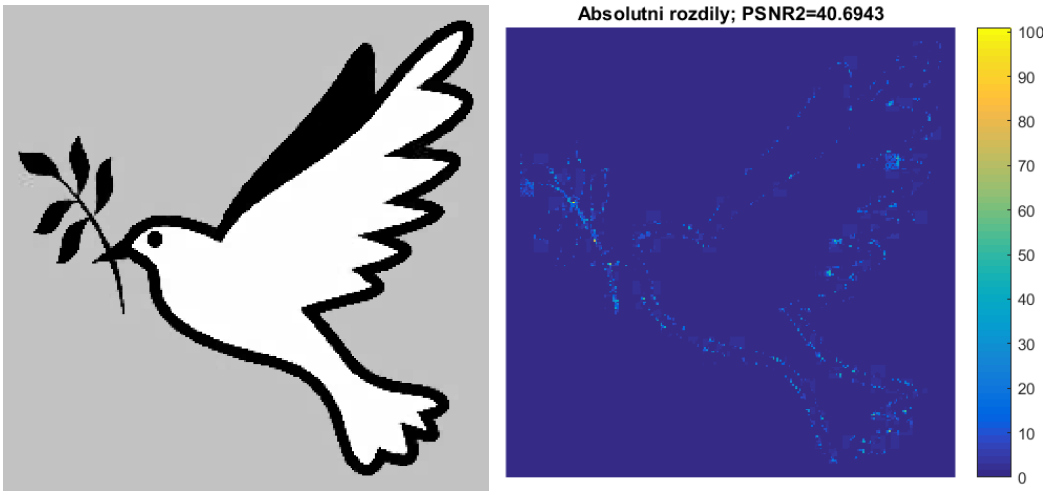
Obrázek 12.5: Srovnání obrázků po rekonstrukci při zachování 25% hranových pixelů.



(a) Výsledek po použití upraveného JPEG algoritmu.

(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

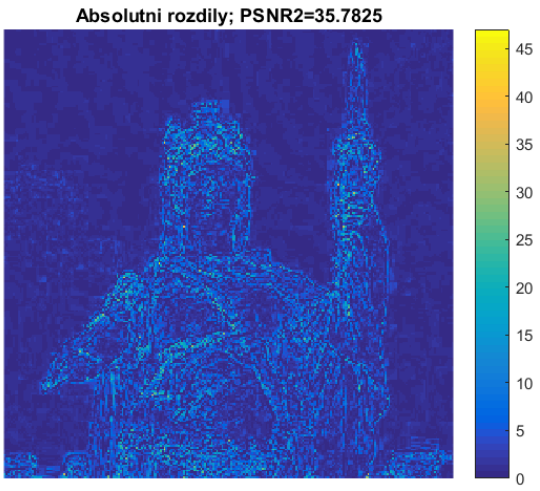
Obrázek 12.6: Srovnání obrázků po rekonstrukci při zachování 25% hranových pixelů.



(a) Výsledek po použití upraveného JPEG algoritmu.

(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

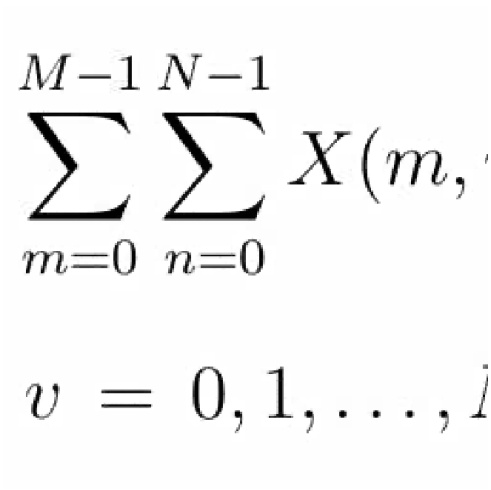
Obrázek 12.7: Srovnání obrázků po rekonstrukci při zachování 75% hranových pixelů.

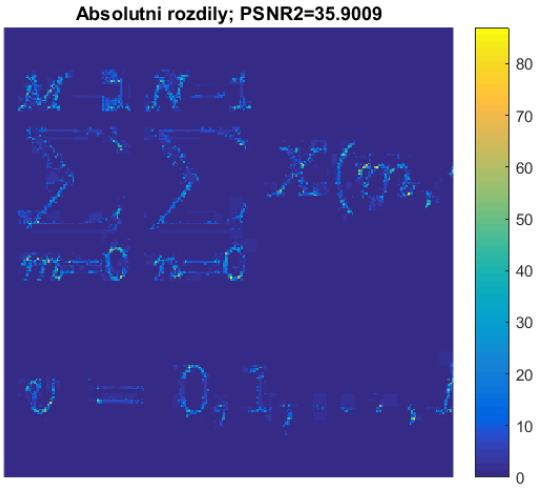


(a) Výsledek po použití upraveného JPEG algoritmu.

(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

Obrázek 12.8: Srovnání obrázků po rekonstrukci při zachování 75% hranových pixelů.



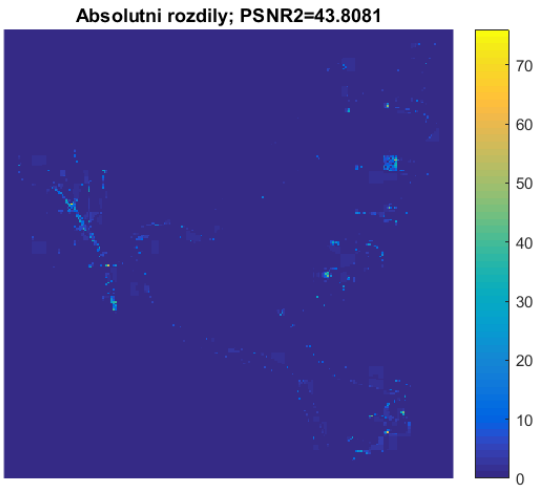


(a) Výsledek po použití upraveného JPEG algoritmu.

(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

Obrázek 12.9: Srovnání obrázků po rekonstrukci při zachování 75% hranových pixelů.

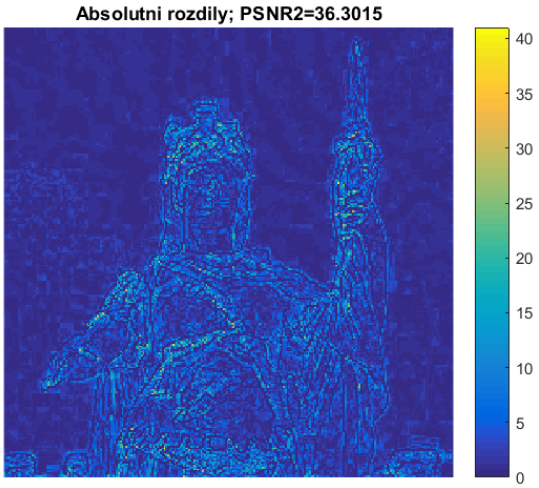
(a) Výsledek po použití upraveného JPEG algoritmu.



(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

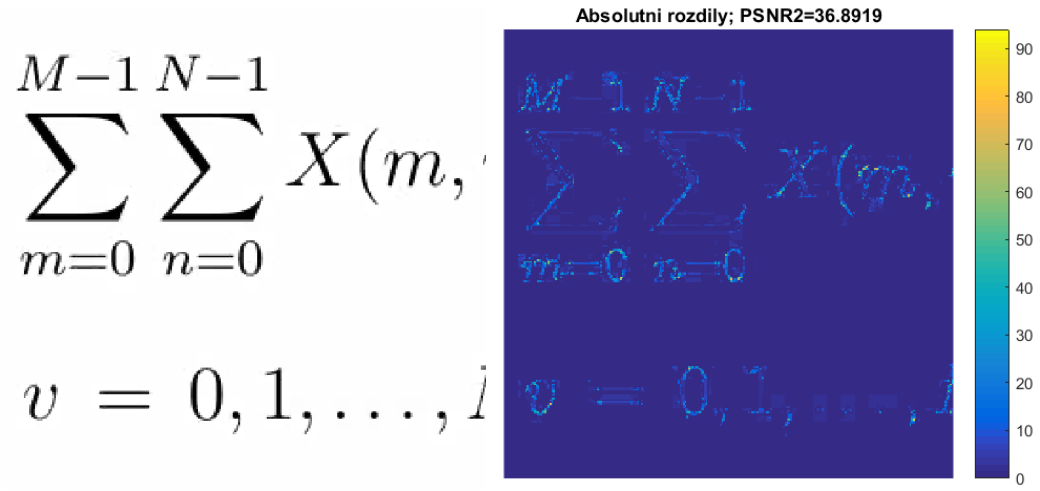
Obrázek 12.10: Srovnání obrázků po rekonstrukci při zachování 100% hranových pixelů.

(a) Výsledek po použití upraveného JPEG algoritmu.



(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

Obrázek 12.11: Srovnání obrázků po rekonstrukci při zachování 100% hranových pixelů.



(a) Výsledek po použití upraveného JPEG algoritmu.

(b) Rozdíl původním obrázkem a výsledkem upraveného algoritmu.

Obrázek 12.12: Srovnání obrázků po rekonstrukci při zachování 100% hranových pixelů.