VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

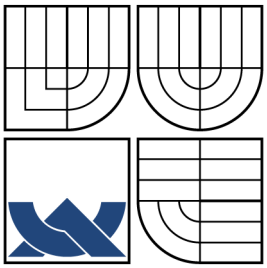
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

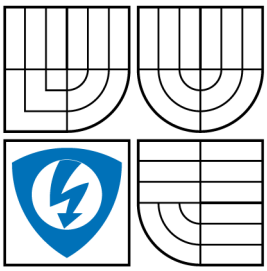
OBRAZOVÉ KODEKY ZALOŽENÉ NA VLNKOVÉ TRANSFORMACI

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE TOMÁŠ KISKA
AUTHOR



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

OBRAZOVÉ KODEKY ZALOŽENÉ NA VLNKOVÉ TRANSFORMACI

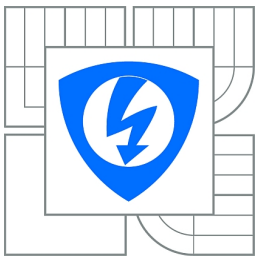
IMAGE CODECS BASED ON THE WAVELET TRANSFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — TOMÁŠ KISKA
AUTHOR

VEDOUCÍ PRÁCE — Mgr. PAVEL RAJMIC, Ph.D.
SUPERVISOR

BRNO 2013



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Bakalářská práce

bakalářský studijní obor
Teleinformatika

Student:	Tomáš Kiska	***ID:***	134520
Ročník:	3	***Akademický rok:***	2012/2013

NÁZEV TÉMATU:

Obrazové kodeky založené na vlnkové transformaci

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte základy vlnkové transformace pro zpracování obrazů. Zejména se zaměřte na kompresi. Seznamte se důkladně s kodeky EZW, SPIHT, EBCOT (JPEG2000) a implementujte je v prostředí Matlab. Porovnejte úspěšnost kodeků pomocí metod pro kvalitatvní i kvantitativní srovnání obrazů (originál/komprimovaný).

DOPORUČENÁ LITERATURA:

[1] William A. Pearlman and Amir Said, Digital Signal Compression: Principles and Practice, Cambridge University Press, 2011.

[2] A. Said and W. A. Pearlman, A new fast and efficient image codec based on set partitioning in hierarchical trees. IEEE Transactions on Circuits and Systems for Video Technology 6 (3): 243–250, 1996

[3] David S. Taubman, Michael W. Marcellin, Jpeg2000:
Image Compression Fundamentals, Standards, and Practice, 2002.

Termín zadání:	11.2.2013	***Termín odevzdání:***	5.6.2013

Vedoucí práce: Mgr. Pavel Rajmic, Ph.D.
Konzultanti bakalářské práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

ABSTRAKT

V této bakalářské práci je popsána problematika obrazových kodeků založených na vlnkové transformaci, která se dnes již běžně používá hlavně při práci s digitálními fotografiemi nebo jinými snímky. Pomocí těchto algoritmů lze komprimovaná data efektivně ukládat.

Jsou zde popsány hlavní obrazové formáty a typy komprese, vhodné pro úpravu snímků. Dále jsme stručně obeznámeni se způsobem hodnocení kvality výsledného snímku. Zaměření je tedy na diskrétní vlnkovou transformaci, při které se využívají tzv. vlnky, z niž nejznámější je Haarova nebo Daubechies.

Dále jsou popsány barevné prostory, ve kterých může být snímek zobrazen. Při kódování a dekódování je to model RGB a YC_BC_R. V neposlední řadě je zmíněn pojem dekompozice obrazu.

Nakonec jsou uvedeny a podrobně rozebrány obrazové kodeky EZW, SPITH a EBCOT.

KLÍČOVÁ SLOVA

EZW, SPITH, EBCOT, komprese, vlnková, DWT, Haar, JPEG2000, kódování, RGB

ABSTRACT

This bachelor´s thesis describes issues of wavelet compression, which is today already in common use mainly at work with digital photographs or other pictures. Using these algorithm you can copy informations and save efficiently.

There are main image formats and types of compression suitable for editing pictures desribed in this thesis. So the focus is on discrete wavelet transform, which uses so-called wavelets, the Haar´s and Daubechies wavelet are the best known of all.

Next chapter of this work explores colour spaces, in which can be a picture displayed. In case of coding and encoding it is model RGB and YC_BC_R. Not least there is mentioned term of picture decomposition.

At the end, there are image codecs EZW, SPITH and EBCOT which are describe in detail.

KEYWORDS

EZW, SPITH, EBCOT, compression, wavelet, DWT, Haar, JPEG, encoding, RGB

KISKA, Tomáš *Obrazové kodeky založené na vlnkové transformaci*: bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2013. 72 s. Vedoucí práce byl Mgr. Pavel Rajmic, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Obrazové kodeky založené na vlnkové transformaci“ jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Mgr. Pavlu Rajmicovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci. A také své přítelkyni Vendule Střeštíkové za podporu a svatou trpělivost.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

ÚVOD

V dnešní době je naprostou samozřejmostí, že si ve svém osobním počítači nebo v televizoru prohlédneme vyfocené fotografie. Avšak jakým principem se obrazová data zobrazují na výsledném snímači je otázka určitých algoritmů kompresních metod. Bez nich bychom si tyto informace ani nedokázali uložit, natož zobrazit. Ve skutečnosti by tato data zabírala neúměrnou diskovou kapacitu a při přenosu by zbytečně zatěžovala počítačové sítě.

Tato práce se zabývá obrazovými kodeky, které vycházejí ze sofistikované a velmi známé kompresní metody založené na vlnkové transformaci obrazu. Tato metoda je moderní záležitostí, která zaznamenala rozvoj v aplikacích v různých oblastech lidské činnosti. Jsou zde popsány zejména všeobecně známé metody a postupy barevných modelů obrazových dat a jejich vzájemné převody, které jsou nad míru důležité k pochopení této práce. Nechybí ani téma Diskrétní vlnkové transformace (DWT), která je základem kompresní techniky, na níž se tato práce zaměřuje.

Na základech DWT jsou pak založeny některé z obrazových kodeků. Pomocí vlnkové analýzy obrazových dat se právě nabízí možnost tato transformovaná data ukládat, a to například pomocí algoritmu EZW (*Embedded Zerotree Wavelet*) nebo SPITH (*Set Partitioning In Hierarchical Trees*), v neposlední řadě pak algoritmu EBCOT, který je součástí známého obrazového formátu JPEG2000, jež podporuje jak zkomprimované, tak nekomprimované ukládání dat. Tento obrazový formát má mnoho nesporných výhod, mezi něž patří především vyšší kvalita obrazu při stejném nebo nižším datovém toku ve srovnání s metodami založenými na diskrétní kosinově transformaci (DCT) obrazu. Další výhodou je absence rušivých blokových artefaktů v obraze, tak typická pro obrazový formát JPEG, který se projevuje zejména při nízkých datových tocích.

Hlavní cílem nejen výše uvedených, ale i ostatních kompresních metod, je dosáhnout vysokého kompresního poměru při zachování poměrně dobré kvality obrazu a nízké náročnosti na výpočetní výkon a čas.

S tímto přímo souvisejí techniky, které posuzují výslednou kvalitu komprimovaného obrazu. Mezi ně patří PSNR, neboli špičkový poměr signálu k šumu, dále pak SSIM index, který se v poslední době výrazně osvědčil jako objektivně posuzující faktor s ohledem na lidské vnímání výsledného obrazu. Nepřímo do těchto technik lze rovněž zařadit počet bitů na pixel (bpp).

Závěr této práce je pak věnován konkrétním, výše zmíněním obrazovým kodekům a jejich principu kódování. Následně jsou tyto metody vyhodnoceny pomocí technik posuzující výslednou kvalitu komprimovaného obrazu.

1 REPREZENTACE OBRAZU V POČÍTAČI

V této kapitole si podrobně představíme, jak se reprezentuje obraz v počítači. Na začátku si přiblížíme některé barevné modely a jejich vzájemné převody, dále si popíšeme paletové a tzv. „True Color“ vyjádření a v závěru konkrétní obrazové formáty, které jsou v dnešní době běžně používané. U těchto formátů si pak blíže srovnáme hlavní výhody a nevýhody formátů JPEG a JPEG2000.

1.1 Barevné modely

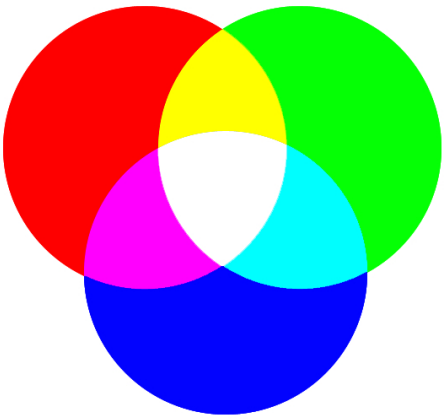
Definicí barevného modelu je spousta. Je možné ho popsat jako abstraktní matematickou strukturu popisující barvy, jako n-tice čísel. Musíme však brát v potaz, že jde o popis základních barev a určující způsob mísení těchto barev do barvy výsledné. V přírodě se vyskytují barvy, jež jsou dány směsicí různých světel vlnových délek, a právě různé druhy barevných modelů se snaží napodobit určitou barvu co nejvěrněji, aby byl výsledný vjem pro lidské oko přirozený. Mezi základní barevné modely patří zejména barevný model RGB, YUV, YC_BC_R nebo třeba CMYK. Model CMYK řeší otázku barev typických pro míchání malířských nebo tiskařských barev, a tudíž není pro naše účely a účely této práce důležitý. Mezi dalšími modely lze zmínit např. HLS, HSV neboli HSB. Tyto barevné modely odpovídají popisu barev, tzv. „*intuitivnímu popisu barev*“, na který je člověk zvyklý. Nyní si pojďme některé barevné modely a jejich převody více přiblížit.

1.1.1 Model RGB

Za nejzákladnější barevný model lze označit právě barevný model RGB [4]. Jeho název vyplývá ze tří základních barevných složek, a to červené (R – red), zelené (G – green) a modré (B – blue).

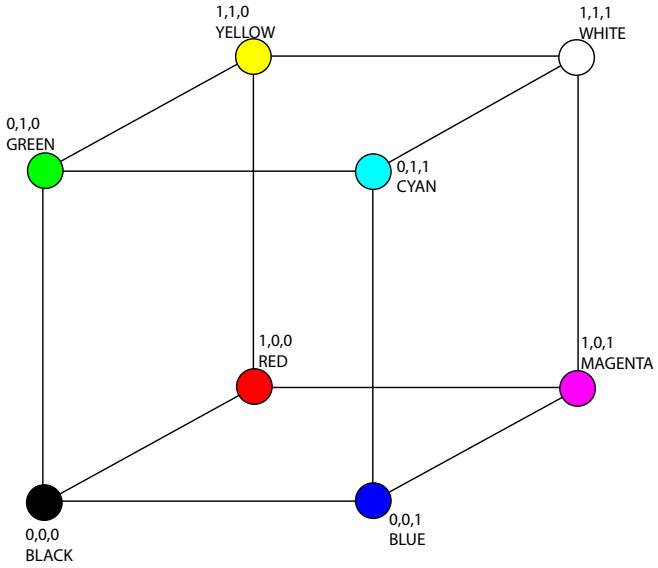
Barevný model RGB je součtový, neboli aditivní (obr. 1.1). Princip spočívá v tom, že si jednotlivé barevné složky můžeme představit jako zářící reflektory vyzařující světlo o daných vlnových délkách. Čím více se barvy překrývají (sčítají), tím jsou světlejší a světlejší. Výsledným sečtením všech barvonosných složek vzniká barva bílá. Tento model lze také vyjádřit pomocí jednotkové krychle souřadnic (obr. 1.2).

Jednotlivé osy souřadnicového systému představují jednotlivé barevné složky barevného prostoru. Na souřadnicích (0,0,0) nalezneme černou barvu, naopak v protilehlém rohu (1,1,1) barvu bílou, na tělesové uhlopříčce pak můžeme vidět stupně šedi. Libovolný bod se souřadnicemi (R,G,B) v této krychli udává hodnotu barvy na intervalu $\langle 0, 1\rangle$. Často se toto zobrazení používá s 24bitovou hloubkou barvy. Hloubce barvy se budeme věnovat v pozdější kapitole. K vyjádření se často používá



Obr. 1.1: Aditivní mísení barev v modelu RGB.

hexadecimální číselná soustava, a to v hodnotách 00 až FF. Toto může být převedeno jako #FF0000, kde každá dvojice představuje hodnotu barvy jednoho kanálu RGB. Využívá se hlavně ve značkovacím jazyce HTML.



Obr. 1.2: Model RGB

1.1.2 Modely YUV a YC_BC_R

Barevný model YUV a YC_BC_R vycházejí z myšlenky oddělit jasovou složku od složek barevných. Model YUV [4] se používá především v televizní technice, kde možnost oddělit složku jasovou řešil především přechod mezi černobílým a barevným vysíláním, kdy v případě černobílého obrazu se zobrazovala pouze jasová složka Y

(luminance). Zbylé dva barevné signály pak doplnily obraz v barevný televizní signál. Barevná složka U je někdy označována jako B-Y, značící barvu modrou a R-Y pak barvu červenou.

Z hlediska zobrazení digitálního obrazu a videa se pak používá barevný model $\mathrm{YC_BC_R}$ [13], který přímo vychází z YUV. V tomto případě má oddělení jasové složky Y význam z hlediska digitální komprese, kde se používá především. V tomto případě je třeba uvést, že tento barevný model se nepoužívá k zobrazení obrazové informace, ale pouze jako prostředek pro kompresi. Výhoda tohoto rozdělení spočívá v tom, že je vhodné kódovat každou z těchto složek zvlášť.

1.1.3 Přepočet barev z RGB do $\mathrm{YC_BC_R}$

Jak již bylo uvedeno výše, barevný model RGB není vhodný pro digitální kompresi, proto se musí převádět (přepočítávat) vhodnými metodami do barevného modelu $\mathrm{YC_BC_R}$. Jasová složka Y, která nám nese informaci o jasu, by se dala vyjádřit pomocí barevných složek RGB modelu vztahem [1]

$$Y = k_\mathrm{r}R + k_\mathrm{g}G + k_\mathrm{b}B, \tag{1.1}$$

kde R, G, B představují jednotlivé barevné složky RGB modelu, Y výslednou složku modelu $\mathrm{YC_BC_R}$ a k pak váhový faktor, který udává hodnoty pro jednotlivé složky $k_\mathrm{r} = 0{,}299$, $k_\mathrm{g} = 0{,}587$ a $k_\mathrm{b} = 0{,}114$, známé z televizního standardu ITU-R BT.601.

Zápis jednotlivých barevných složek je pak vyjádřen jako rozdíl barvy a jasové složky

$$C_\mathrm{r} = R - Y, \tag{1.2}$$

$$C_\mathrm{g} = G - Y, \tag{1.3}$$

$$C_\mathrm{b} = B - Y. \tag{1.4}$$

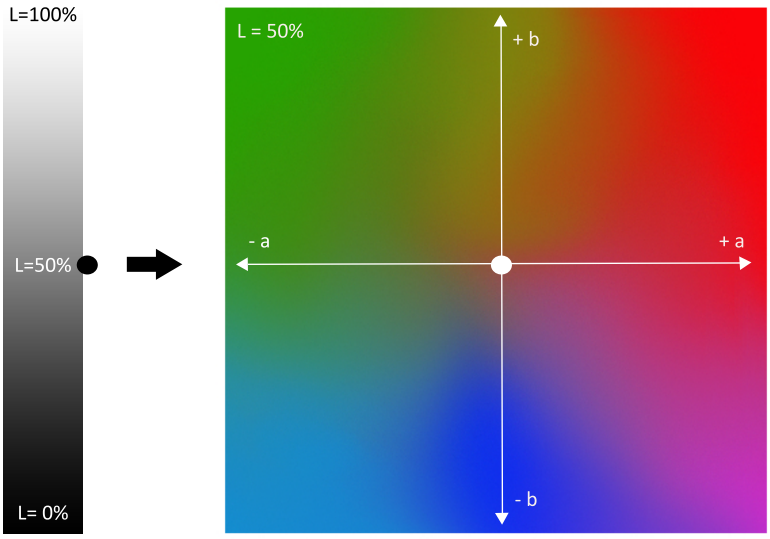
Z podstaty modelu $\mathrm{YC_BC_R}$ vyplývá, že jsou zde zastoupeny pouze dvě barevné složky. Je to dáno především tím, že lidské oko není tolik citlivé na barvy, jako na jas. Tento model je zmíněné skutečnosti přizpůsoben, a proto dává větší význam jasu v podobě samostatné složky Y. Samotný proces přepočítání (transformace) lze vyjádřit maticovým uspořádáním [17]

$$\begin{pmatrix} Y \\ C_\mathrm{b} \\ C_\mathrm{r} \end{pmatrix} = \begin{pmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}169 & -0{,}331 & 0{,}5 \\ 0{,}5 & -0{,}419 & -0{,}0813 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix}. \tag{1.5}$$

Hodnoty v první řádku matice představují konkrétní váhové faktory k pro jednotlivé barevné složky. První modrý barevný (chrominanční) signál C_B, je vyjádřen rozdílností mezi původní modrou složkou a novou jasovou složkou.

1.1.4 Prostor L*a*b* (CIELAB)

Barevný prostor L*a*b*, zvaný také jako CIELAB [13], se dá charakterizovat jako prostor obsahující všechny barvy, které lze lidským zrakem zpozorovat. Mezi nesporné výhody patří také jeho nezávislost na zařízení, protože popisuje spíše jak barva vypadá, než jaké hodnoty barevných složek výsledná barva obsahuje. Prostor je popsán třemi složkami a podobně jako je tomu u modelů YUV nebo YC_BC_R, i zde hraje roli jasová složka, tentokrát pod označením L (Lightness). Zbývající dvě složky popisují přechody barev, složka a popisuje přechod barev zelená – červená a složka b pak modrá – žlutá. Podrobně tyto vlastnosti můžeme vidět na obr. 1.3 kde je názorně zobrazen prostor LAB pro danou hodnotu jasu L. V našem případě složku L (světlost bodu) zobrazuje levý sloupec v hodnotách 0 až 100 %, tj. hodnota 0 % udává černý bod a naopak hodnota 100 % pak bod bílý. Dále z obrázku je patrné zobrazení barevného prostoru o řezu úrovně složky L = 50 %. Osa a zde popisuje barvu bodu od barvy zeleno-modré (záporné hodnoty) po barvu červeno-purpurovou (kladné hodnoty). Naopak osa b zobrazuje barvu od modro-purpurové v záporných hodnotách po zeleno-žluto-červenou v kladných hodnotách. Tento typ zobrazení je známý z programu Adobe Photoshop, v němž je možno zadávat hodnoty od -128 do $+127$.



Obr. 1.3: Názorné zobrazení prostoru LAB.

Pokud vezmeme v potaz barevnou odchylku [13], která udává barevný soulad či nesoulad barev (značena symbolem ΔE), můžeme tuto odchylku v prostoru LAB vyjádřit jako

$$\Delta E = \sqrt{(L - \tilde{L})^2 + (a - \tilde{a})^2 + (b - \tilde{b})^2}. \tag{1.6}$$

Je známo, že člověk dokáže rozeznat v některých odstínech i malou změnu a z vyjádření je patrné, že tento výpočet vůbec nebere v úvahu citlivost zraku. Pro přesnější porovnávání se využívá barevná odchylka zvaná $\Delta E\,2000$, která tuto v potaz už bere.

1.2 Paletové vs. True Color vyjádření

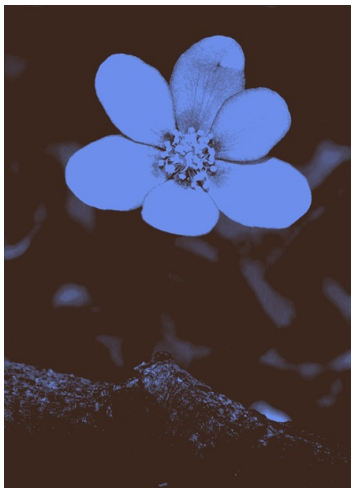
V této části si popíšeme hlavní rozdíly mezi paletovým vyjádřením barvy a vyjádřením tzv. True color, které vycházejí z barevné hloubky obrazu. Dále si vysvětlíme, co vyjadřuje indexovaná barevná paleta a předvedeme si názorné zobrazení jednotlivých vyjádření.

1.2.1 True Color

Abychom mohli popsat True color vyjádření, zavedeme si pojem barevná hloubka obrazu, která popisuje, kolik bitů je použito k popisu určité barvy nebo pixelu v obrazu. Hodně často se toto vyjádření dá shrnout jako známé: „počet bitů na pixel (bit per pixel – bpp)“.

Barevné hloubky

Mějme jednobitovou barevnou hloubku ($2^1 = 2$ barvy), což znamená že máme k dispozici monochromatické zobrazení, které většinou obsahuje barvu bílou a černou. Pro ukládání nebo zobrazení obrazových dat se používá především barevný model RGB. Někdy se pro průhlednost používá rovněž přídavný kanál zvaný A (alfa kanál), který nese název RGBA. Počet bitů a počet výsledných barev můžeme vidět v tab. 1.1. Důvodem, proč jsou u 8bitové a 16bitové hloubky bity rozděleny nerovnoměrně, je skutečnost, že lidské oko je nejméně citlivé na barvu modrou, o něco více pak na barvu červenou. Zelená barva je pro lidské oko nejcitlivější, proto je v tomto případě dominantní. Barva označovaná jako True color má 24-bit hloubku, což nám značí, že máme konkrétně v barevném modelu RGB 8 bitů na každou barvu. V uvedeném modelu se nachází velmi velké množství barev, které může být zobrazeno v široké paletě odstínů a tudíž vytváří vysoce kvalitní obrazy (fotografie, detailní grafické uspořádání). Z uvedeného vyplývá, že máme k dispozici 256 odstínů červené, zelené a modré barvy, což nám dává dohromady 16 777 216 různých barev. Pokud si uvědomíme, že lidské oko dokáže rozpoznat kolem deseti miliónu barev, je zobrazení True color více než dostatečné, a proto je název True color (opravdové barvy) zcela výstižný.



(a) 1bitová

(b) 2bitová

(c) 4bitová

(d) 6bitová

(e) 8bitová

(f) True color 24bitová

Obr. 1.4: Ukázka barevné hloubky 1 až 24bitové

Tab. 1.1: Rozložení stupně bitů v jednotlivých kanálech RGBA.

Barevná hloubka	**bity na kanál**			
Počet bitů	R	G	B	A
8bit	3	3	2	
16bit	5	6	5	
18bit	6	6	6	
24bit	8	8	8	
32bit	8	8	8	8

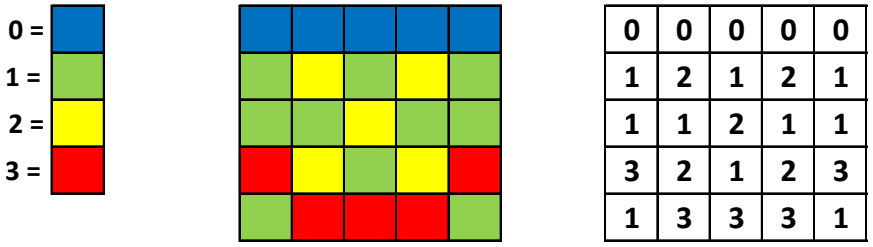
Srovnáním obr. 1.4 můžeme pozorovat, jaký vliv má barevná hloubka na výsledný vjem. Na obr. 1.4a vidíme ukázku o barevné hloubce 1 bit obsahující právě dvě barvy. V uvedeném případě se jedná o dvě nejvíce zastoupené barvy z originálu obr. 1.4f s barvami True color (24 bitů). Z ukázkového příkladu je patrné, že postupným zvyšováním barevné hloubky získává ukázková fotografie přirozenější barevnou hloubku, na níž je lidské oko mnohem lépe přizpůsobeno. Z těchto poznatků je dále patrný význam vyšší barevné hloubky u fotografií nebo co do barev složitějších obrazů.

1.2.2 Barevná paleta

Stejným způsobem, jako vyjádření barvy pomocí barevné hloubky, můžeme vyjádřit barvu i pomocí barevné palety. Barevná paleta by se dala definovat jako matice čísel, kde každému číslu v této matici odpovídá jiná barva, tj. každý pixel v obrazu uchovává číslo, které se odkazuje na odpovídající barvu v paletě (matici čísel). Mezi hlavní výhody oproti vyjádření hloubkovému je zde relativně malá paměťová náročnost. Na druhou stranu je třeba uvést, že tato metoda je vhodná pouze pro obrazy s malým počtem různých barev, tudíž s tím související malé barevné rozlišení výsledného obrázku a nevhodnost užití pro detailnější, barevně bohatější, obrazy. Na obr. 1.5 můžeme vidět příklad 2bitového ($2^2 = 4$ barvy) obrázku s indexem barev. Každá barva každého pixelu je určena číslem, které odpovídá barvě v paletě. Velikost palety se většinou používá do 8bitové hloubky tedy 256 barev.

1.3 Příklady konkrétních formátů

V úvodu této části si představíme obrazové formáty, které jsou v dnešní době nejvíce využívány. Setkáváme se s nimi téměř každý den na Internetu nebo u digitálních fotografií v počítači. Rozebereme si základní poznatky obrazových formátů GIF,



Obr. 1.5: Příklad palety a indexovaných barev.

BMP a PNG [9], v neposlední řadě si probereme blíže formáty JPEG a JPEG 2000. Konkrétně formát JPEG 2000 přímo z vlnkové transformace vychází.

1.3.1 GIF, PNG a BMP

GIF

Formát GIF (*Graphics Interchange Format*) byl vyvinut v roce 1987 americkou společností CmpuServe. Používá bezztrátovou komprimační metodu, která funguje na principu ukládání jednotlivých řádků pomocí čísel, jež odpovídají barevné paletě známé z předchozí kapitoly. Maximální počet barev pro tento formát byl stanoven na 256, a tudíž se jedná o 8bitovou hloubku. Velikost uloženého souboru je přímo úměrná počtu barev, kterými je daný obrázek uložen. Nejmenší velikosti dosáhneme tím, že obraz bude obsahovat co nejmenší počet barev. Z těchto poznatků je zřejmé, že tento obrazový formát rozhodně není vhodný pro ukládání fotografií či kresbě, ale spíše najde uplatnění ve tvorbě webové grafiky v podobě různých tlačítek, ikon apod.

BMP

Obrazový formát BMP (*Windows Bitmap*), který byl prvně představen v r. 1988, je typickým příkladem bitmapových grafických formátů. Mezi hlavní výhody tohoto formátu patří zejména jeho extrémní jednoduchost a dokumentovanost. Obraz ve formátu BMP bývá ukládán po jednotlivých pixelech, čímž využívá vlastností barevné hloubky a počtu bitů na pixel., navíc mohou pro 8bitové obrázky používat stupně šedi pro černobílou reprodukci obrazu v 256 odstínech šedi. Za hlavní nevýhodu tohoto formátu však považuji absenci možnosti využití, byť jen minimální, komprese obrazu, z čehož vyplývá, že obrazy uložené v BMP jsou co do velikosti dat mnohem větší, než obrazy stejného rozměru uložené v jiném obrazovém formátu, které využívají kompresní metody. Z vysokých nároků na uložení jasně vyplývá, že užití tohoto formátu na internetu je zcela nevhodné.

PNG

Tento obrazový formát (*Portable Network Graphic*) byl vyvinut jako zdokonalení formátu GIF. Jak už název napovídá, byl především navržen pro internetovou grafiku. Podporuje barevnou hloubku od 2 do 48 bitů a kanál pro průhlednost s 256 úrovněmi. Obrázky, podobně jako ve formátu GIF, se ukládají pomocí komprimačního algoritmu (*Deflate*) jako bezeztrátové. Tento formát má oproti svému předchůdci i některé nevýhody, typickým příkladem je skutečnost, že prakticky nepodporuje jednoduché animace.

1.3.2 Srovnání JPEG a JPEG 2000

Tato část je věnována známému obrazovému formátu JPEG a novějšímu formátu JPEG2000, který vychází ze stejné myšlenky, ale využívá jiných metod pro ukládání a zpracování obrazu.

JPEG

Formát JPEG [17] je jedním z nejrozšířenějších formátů pro ztrátové ukládání fotografií v počítači. Zkratka tohoto formátu vychází z JFIF (JPEG *File Interchange Format*)), přímo zkratka JPEG značí název skupiny, která jej vytvořila (*Joint Photographic Experts Group*). Uvedený formát, podporující 24bitovou hloubku, jež představuje 16 777 216 barev, ukládá obraz pomocí RGB modelu.

Typickým příkladem použití jsou fotografie uložené na Internetu. Díky jeho velké barevné hloubce je pro tyto účely více než dostatečný. Není naopak vhodný k ukládání textu, ikon, perokresby, kde kompresní metoda tohoto formátu tvoří rušivé elementy, zvané blokové artefakty. Pro tyto účely jsou vhodnější formáty PNG nebo GIF. Nesporná výhoda tohoto formátu spočívá v jeho kompresi, kde využívá nedokonalosti lidského oka, tudíž funguje mnohem lépe než bezeztrátové metody při stále velmi dobré kvalitě výsledného obrazu.

JPEG2000

Postupem doby byl vyvinut nový obrazový formát, který měl nahradit stávající JPEG. Tímto formátem je myšlen obrazový formát JPEG2000 [17], který byl poprvé představen v roce 1995. Název samotného formátu pak značí tehdejší předpokládaný rok uvedení. Na jeho vývoji spolupracovala jak známá skupina *Joint Picture Expert Group*, tak skupiny nazvané ISO JPEG *Commitee* a *Digital Imaging Group*. V roce 2000 se na svět podívala první specifikace a tento formát se stal otevřeným stejně jako JPEG. Typické pro soubory JPEG200 jsou koncovky .jp2 .j2k, .jpf, .jpx, .jpm, .mj2.

Tento obrazový formát je jedním z těch, které používají pro kompresi obrazových dat vlnkovou transformaci, přesněji její diskrétní formu (DWT). Další zajímavostí je, že dokáže ukládat jak ztrátově, tak bezztrátově za použití stejného kodéru, s čímž úzce souvisí rychlejší a kvalitnější komprese dat oproti stávajícímu JPEG. Rozlišení není vůbec limitováno a na rozdíl od JPEG může přesáhnout 64000×64000 pixelů. Za přínos lze považovat i kvalitnější zpracování ostrých přechodů v obrázku. Novinkou u formátu JPEG2000 je, že kromě obrazové informace může obsahovat také metadata, která tvoří doplňující informaci o obrázku, jako jsou např. název, datum a čas pořízení,autor, popis, vlastník autorských práv apod.

Srovnání

Nyní, když jsme si představili oba formáty, si názorně převedeme jejich vzájemné výhody či nevýhody. Jako názorný příklad jsme použili obr. 1.6, na kterém demonstrujeme oba obrazové formáty [9]. Největším rozdílem těchto formátů je, že JPEG2000 používá pro svou kompresi diskrétní vlkovou transformaci DWT, místo diskrétní kosinovy transformace (DCT) typické pro JPEG. Kvůli rozdílným přístupům nám na obrázcích vznikly dva různé výsledky při relativně stejné velikosti souboru. Na obr. 1.6c vidíme, že se při vyšší kompresi objevují rušivé elementy v podobě tzv. blokových artefaktů, které jsou tolik typické pro JPEG. Důsledek těchto artefaktů je dělení bloků na dlaždice 8×8, s nimiž pracuje DCT. Naopak u JPEG2000 (obr. 1.6b) využívá DWT, která více přistupuje k ostrým hranám přechodu a při větší kompresi se zdánlivě rozmazávají. Tento efekt je však pro lidské oko mnohem příjemnější a hlavně přirozený. Z uvedeného plyne, že pomocí JPEG2000 můžeme při zachování relativně dobře pozorovatelného obrazu dosáhnout větších kompresních poměrů oproti formátu JPEG.

V neposlední řadě je nutno podotknout, že DWT používaná v JPEG2000 má mnohem vyšší paměťovou náročnost, a proto se tento obrazový formát nestal tolik rozšířeným jako klasický JPEG.

(a) Originál (112 kB)

(b) JPEG2000 (41,3 kB)

(c) JPEG (40,4 kB)

Obr. 1.6: Srovnání JPEG a JPEG2000 oproti originálu, při uvedené velikosti souboru.

2 KOMPRESE OBRAZŮ

S přelomem nového tisíciletí jsme byli svědky mnoha změn, někteří z nás dokonce hovoří o revoluci, na cestách komunikace a vývoje. Tyto změny se dokonce odehrávají i v přítomnosti, o čemž napovídá například rostoucí význam Internetu, jako masivního zdroje komunikace, a rovněž zvyšující se důležitost videokomunikace. Datová komprese je jedna z technologií, která umožňuje zkoumat všechny tyto aspekty multimediálních komunikací. Nemusí být však praktické přidávat obrázky, natož audio a video, na webové stránky, pokud není předem proveden algoritmus datové komprese. Mobilní telefony pak nemusí být schopny poskytování komunikace s dostačující čistotou. Příchod digitální televize by bez komprese nebyl možný. Datová komprese, která byla po dlouhou dobu doménou relativně malé skupiny vědců a vývojářů, se nyní stává všudypřítomnou. Používání faxového přístroje, modemu, či satelitní televize vás dostává do kontaktu s kompresí.

Komprese obrazových dat je zpracování těchto dat za pomocí určitých metod za účelem zmenšení jejich objemu (velikosti souboru) při současném zachování informací v datech obsažených. Dalším cílem komprese dat je zmenšit datový tok přes přenosová média a zároveň zmenšit nároky zdrojů informace na ukládání. Z toho plyne že pokud budu chtít přenést přes počítačovou síť v niž je omezena přenosová rychlost, tak velikost některých dat mohou tuto maximální přenosovou rychlost mnohonásobně převyšovat. Řešení tohoto problému se právě naskýtá v perspektivních možnostech datové komprese.

2.1 Bezeztrátová komprese

Technika bezeztrátové komprese [16], jak už sám název vystihuje, zahrnuje informace, které nejsou ovlivněny ztrátami. Pokud mají byt data bezeztrátově zachyceny, původní data musí být přesně obnovena z dat zkomprimovaných. Bezeztrátová komprese je obecně používaná pro aplikace, které netolerují žádné rozdíly mezi původními a obnovenými daty.

Důležitou součástí samotné komprese jsou kódovací algoritmy. V bezeztrátové kompresi se často používá několik těchto algoritmů najednou. U některých komprimacích jsou data nejdříve transformována a až poté kódována. Konkrétně transformace se používá za účelem dosažení lepších kompresních poměrů. Nyní si přiblížíme některé z těchto kódovacích algoritmů.

2.1.1 Run-Length Encoding

Mezi základní kódovací algoritmy lze zařadit právě Run-Length Encoding nebo zkráceně RLE. Jak už název napovídá jedná se o algoritmus, který pracuje s délkou posloupnosti a její hodnotou. Algoritmus kóduje vstupní data tím že kóduje posloupnosti stejných hodnot do dvojic. Tudíž účinnost této komprese spočívá v četnosti stejných znaků, kódovaného obsahu, která musí obsahovat delší sekvence stejných znaků, pokud tomu tak není, účinnost komprese výrazně klesá a v extrémních případech dokonce až dvojnásobná velikost výstupních dat.

Mějme příklad, kde máme vstupní tok kodéru RLE znaky:

AAAAAXXAAAAAAAAAAXAAAAAXX.

Tyto znaky pak na výstupu kodéru vypadají následovně:

5A2X11A1X5A2X.

Nejčastějším případem je komprese bytového proudu. V tomto případě se to realizuje pomocí dvojic bytů, kde první byte určuje hodnotu kódovaného znaku a ten druhý pak četnost opakování. Mez hlavní nevýhodu patří to, že pokud se na vstupu kodéru vyskytují pouze jedinečné znaky, tak na výstupu se kódují pomocí dvou bytů.

Na příkladu mějme tok jedinečných znaků:

ABCDEFGHIJK.

Tyto znaky pak na výstupu kodéru tvoří dvojnásobnou velikost:

1A1B1C1D1E1F1G1H1I1J1K.

Použití tohoto kódování v počítačové grafice při kompresi obrazu, se využívá v případě, když jsou v obrazu patrné velké plochy o stejné hodnotě barvy. Naopak při použití tohoto kódování na fotografiích by bylo zcela nevhodné. Často se tato metoda používá jako pomocná ve formátu JPEG.

2.1.2 Huffmanovo kódování

Dalším kódováním užívaným u bezeztrátová komprese je velmi známé Huffmanovo kódování [10]. Toto kódování využívá bitových řetězců různé délky. Znaky, které s vyskytují nejčastěji jsou převedeny do bitových řetězců s nejkratší délkou (může být vyjádřeno i jedním bitem). Naopak pokud máme na vstup kodéru znaky, které se v kódované zprávě vyskytuji velmi zřídka, jsou převedeny do delších řetězců (dlouhé několik bitů).

Samotná komprese dat probíhá ve dvou krocích. Kodérem projde zpráva(soubor) a vytvoří vytvoří si statistiku četnosti každého znaku. Poté za pomocí těchto četností znaků vytvoří tzv. binární strom a poté následuje samotná komprese. Opačný proces dekomprese se provádí pomocí vytvořeného binárního stromu a postupně dekóduje řetězce proměnlivé délky.

Nechť máme slovo, které se skládá ze čtyř různých symbolů, které označíme jako s_1, s_2, s_3, s_4 (dolní indexy zde značí počet zpráv). Dále jejich pravděpodobnosti výskytu jsou $p_1 = 0{,}04$; $p_2 = 0{,}6$; $p_3 = 0{,}1$; $p_4 = 0{,}26$ Tyto zprávy seřadíme podle jejich pravděpodobnosti výskytu aby platilo:

$$p_1 \geq p_2 \geq p_3 \ldots p_q. \tag{2.1}$$

V našem případě je pořadí následující: s_2, s_4, s_3, s_1. Dalším krokem je to, že sečteme poslední dvě pravděpodobnosti ($p_3 + p_1 = 0{,}14$) a výsledek zařadíme podle velikosti pravděpodobnosti mezi ostatní pravděpodobnosti do druhého sloupce (kroku). Znovu provedeme součet posledních dvou pravděpodobností($p_1 3 + p_4 = 0{,}4$) a tento výsledek opět zařadíme podle velikosti do následujícího slupce. Toto sčítání pravděpodobností provádíme tak dlouho, dokud není součet dvou posledních pravděpodobností ve sloupci roven 1. Posledním dvěma znakům v každém sloupci každého kroku přiřadíme kódové znaky 1 (znak s vyšší pravděpodobností) a 0 (znak s nižší pravděpodobností).Výsledný kód pak sestavím ze znaků 1 a 0 podle toho, jak se daný znak seskupoval s ostatními znaky. Vše je názorně předvedeno na tab. 2.1.

Tab. 2.1: Ukázka Huffmanova kódování

s_i	p_i	I	s_i	p_i	II	s_i	p_i	III
s_2	0,6		s_2	0,6		s_2	0,6	1
s_4	0,26		s_4	0,26	1	s_{134}	0,4	0
s_3	0,1	1	s_{13}	0,14	0			
s_1	0,04	0						

$$s_1(0{,}04) \rightarrow 000;\ s_2 = (0{,}6) \rightarrow 1;\ s_3 = (0{,}1) \rightarrow 100;\ s_4 = (0{,}26) \rightarrow 10$$

2.1.3 Aritmetické kódování

Aritmetické kódování je metoda pro bezztrátovou kompresi dat. Toto kódování podobně jako Huffmanovo kódování je forma entropického kódování s proměnlivou délkou kódového slova. Převádí řetězce do jiného tvaru tím způsobem, že převede

často vyskytující se znaky na menší počet bitů a pro vzácnější znaky pak naopak vetší počet bitů. Tudíž stejně jako Huffmanovo kódování s cílem zabrat co nejméně bitů. Aritmetické kódování pracuje na principu že zakóduje celé vstupní slovo do jednoho čísla nebo zlomku n. Kde pro n platí: $0 \leq n < 1$.

Pro příklad tohoto kódování si vezmeme posloupnost symbolů, které zakódujeme. Mějme posloupnost tří znaků X,Y,Z. Každý z těchto z těchto znaků nese stejnou pravděpodobnost výskytu neboli $p_X = p_Y = p_Z$. Pokud bychom použili jednoduché blokové kódování (např. RLE) tak by zabralo 2 bity na znak, což by bylo plýtvání. U aritmetického kódování reprezentujeme posloupnost jako racionální číslo na intervalu 0 až 2 o základu 3. Každý použitý znak zde představuje jedno číslo 0, 1, 2. Dále mějme posloupnost znaků „XYYZXY“ a z této posloupnosti se stane $0{,}011201_3$. Jelikož máme kód v trojkové soustavě, tak jej převedeme do dvojkové, aby byla zpráva vhodná pro přenos. Převodem nám vznikne $0{,}001011001_2$, cože je pouhých 9 bitů, kterých je o 25% méně než při běžném blokovém kódování. Tento algoritmus je proto velmi vhodný pro dlouhé posloupnosti, neboť nejsme limitováni počtem míst za destinou čárkou. Při opačné procesu dekódování víme že řetězec měl délku 6 znaků, tak můžeme převést číslo jednoduše z trojkové soustavy a zaokrouhlit na 6 destiných míst a odtud máme opět počáteční posloupnost „XYYZXY“.

2.2 Ztrátová komprese

Ztrátová komprese [16] je druhým typem datové komprese digitálních dat. Za pomoci určitých algoritmů zmenšuje objem (velikost) dat a to až na zlomek jejich původní velikosti. Vedlejším efektem tohoto procesu se ztrácejí méně důležité informace a komprimovaných dat je již nelze zrekonstruovat.

I když při ztrátové kompresi dochází ke ztrátě informace nenávratně, tak i přes to je tento způsob ukládání výhodný. Tato skutečnost je odůvodněna velmi výrazným zmenšením dat a s tím související malé nároky na ukládání.

Z podstaty ztrátové komprese plyne že je vhodná právě pro ukládání obrazových a zvukových dat, kdy se využívá nedokonalosti lidských smyslů. Všeobecně je pak tato komprese nepoužitelná v případech, kde je důležité uchovat všechny data a nelze z nich žádné informace odstranit aniž by ztratila význam. Mezi ně lze zařadit text knihy, výsledky měření nebo nějaká aplikace.

2.2.1 Transformační kódování

Obecně platí, že ztrátová komprese může být chápána jako aplikace transformačního kódování, a to v případě používání multimediálních dat a percepčního kódování, kdy

je tato transformace v podstatě přeměna hrubých dat na domény, které tak věrněji odráží skutečný obsah informací.

Právě k transformaci dat původních dat se používají ortonormální transformace. Mezi tyto transformace lze právě zařadit například diskrétní Fourierova transformace (především její rychlá forma FFT – rychlá Fourierova transformace), DCT (diskrétní kosinová transformace) nebo právě DWT (diskrétní vlnková transformace). Tyto transformace převádí původní data do jiných domén (např. z časové oblasti do frekvenční). Proto je většina informací poté uložena s menším objemem (velikostí), než původně. Navíc pokud zbytek dat nahradíme předem známými vypočtenými daty, tak poté data se po zpětné transformaci budou velmi podobat datům původním.

Transformace pracující z celočíselnými hodnotami pracují obvykle o něco hůře, ale jsou snadněji implementovatelné. Naopak transformace pracující zaokrouhlováním reálných čísel nemá ztrátu dat tak velkou a obvykle celkový výsledek předčí transformaci pracující s reálnými čísly. Pokud počítáme s tím že budou potlačena některá data je vhodné zvolit ztrátovou transformační metodu.

Rychlá Fourierova transformace (FFT)

Rychlá Fourierova transformace (Fast Fourier transform – FFT) je efektivní algoritmus pro výpočet diskrétní Fourierovy transformace (DFT) i její inverzní verze. Právě FFT je velmi známá z digitálního zpracování signálu v našem případě digitalizovaného 2D signálu v podobě obrazové informace.

Základním princip FFT je právě rozdělení posloupnosti vzorků na dvě posloupnosti. Jedna obsahuje všechny liché vzorky a ta druhá liché vzorky diskrétního signálu DFT. Aby posloupnosti měli stejnou velikost, tak musí být počet všech prvků sudý.

Diskrétní kosinová transformace (DCT)

Diskrétní kosinová transformace (Discrete Cosine Transform – DCT) [8] je další transformace z řad transformačního kódování. Jedná se o novější sofistikovanější metodu pro zpracování obrazu než FFT. Právě obraz v tomto případě můžeme považovat za dvourozměrný vzorkovaný signál. Nalezením korelace blízkými nebo i vzdálenějšími obrazovými body (pixely) vzniká tzv. mezipixelová redundance.

Při transformaci se provádí převod zpracovaného signálu z časové oblasti do oblasti frekvenční. Důvodem je předpoklad, že právě obrazy reálných předmětů (např. na fotografii) neobsahují velké množství energie ve vyšších frekvencích a tudíž je vhodné shromáždit co největší množství dat do malého počtu koeficientů. Důsledkem tohoto kódování by mělo být snížení počtů bitů nesoucí viditelnou informaci, jedná se o tzv. psychovizuální redundanci.

Pro zpracování dvourozměrného obrazu se využívá 2D forma DCT, která vzniká z zobecněním 1D DCT. Mějme obraz jako vzorkovaný signál(rastrový obrázek), který máme na vstupu 2D DCT. Tento vstup označíme jako $s(m, n)$. Na výstupu je opět signál jako matice čísel (rastrový obrázek) o hodnotách $t(i, j)$. Velikost 2D DCT nám udává velikost obrázku $M \times N$, která většinou bývá $M = N$. Splnění této podmínky rovnosti vede k rychlejším a jednodušším výpočtům. Výpočet 2D DCT v našem případě dostává tvar:

$$t(i,j) = c(i,j) \sum_{m=0}^{M-1} \sum_{n=1}^{N-1} s(m,n) \cos \frac{\pi(2m+1)i}{2M} \cos \frac{\pi(2n+1)j}{2N} \tag{2.2}$$

Diskrétní vlnková transformace (DWT)

Diskrétní vlnková transformace (Discrete Wavelet Transform – DWT) [14] a její užití při digitálním zpracování obrazu je obrovským přínosem. Dovoluje nám redundanci dat obrazu na úkor vyšších frekvenčních složek a to za minimálního snížení rozlišení. Právě DWT využívá samotný fakt lidského vidění, které jak je známo citlivé více na hrany v obraze než na jeho okolí. Hrany v obraze reprezentují ve spektrální oblasti vyšší frekvence a pokud bychom tyto frekvence odstranili dojde pouze k rozostření hran v obraze a s tím související zanedbatelné zhoršení kvality. Navíc pro lidské oko přijatelnější něž u DCT v podobě blokových artefaktů. Díky této vlastnosti můžeme pomocí DWT dosáhnout většího stupně komprese obrazových dat. Podrobněji se DWT budeme věnovat ve třetí kapitole této práce.

2.3 Způsob hodnocení kvality

Z kompresí obrazových dat úzce souvisí právě způsob hodnocení výsledných dat. V našem případě těch obrazových. Tato problematika je velmi důležitá, neboť samotný výsledek komprese je nejdůležitější faktor a není jej možné zanedbat. Proto zavádíme některé způsoby, které nám dávají určitý nadhled jak výsledek dopadl a jestli je určitá komprese vhodná. Mezi tyto způsoby hodnocení lze zařadit právě samotný kompresní poměr a počet bitů na pixel. V neposlední řadě efektivnější měřítko v podobě PSNR nebo SSIM indexu.

2.3.1 Kompresní poměr a počet bitů na pixel (bpp)

Kompresní poměr by se dal definovat jako podíl velikosti původních (vstupních) ku velikosti dat komprimovaných (výstupních). Mějme obrázek o velikosti 20 MB, ten zkomprimuje a výsledkem bude obrázek o velikosti 4 MB. Z toho plyne poměr

$20/4 = 10/2 = 5$ a tudíž výsledný poměr je $5:1$, tedy pětkrát menší velikost oproti originálu nebo také 80 % úspora místa.

Dalším hodnotícím faktorem je tzv. bpp (bit per pixel), které značí počet bitů na pixel v obraze. Znamená to kolik průměrně bitů vychází na jeden obrazový bod(pixel) v obraze. Výpočet by se dal popsat jako podíl počtů bitů na jeden pixel původního obrazu a násobku velikosti původního obrazu vzhledem k zakódovanému.

2.3.2 PSNR (Špičkový poměr signálu k šumu)

Společně s kompresním poměrem a počtem bitů na pixel lze zařadit Špičkový poměr signálu k šumu zvaný také zkráceně jako PSNR (Peak Signal to Noise Ratio) [18]. Tato hodnota nám značí poměr mezi maximální možnou energií signálu a energií šumu. Jelikož spousta signálu má velmi široké a dynamické spektrum, velmi často se PSNR vyjadřuje v logaritmickém měřítku a jeho jednotkou jsou decibely dB.

Proto abychom mohli definovat PSNR je třeba znát střední kvadratickou chybu (mean squared error), kterou budeme označovat jako zkratku anglického názvu jednoduše MSE. Poté MSE je definována pro dva monochromatické (černobílé) obrazy I a K o rozměrech $m \times n$ jako:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} ||I\ (i,j) - K\ (i,j)||^2. \tag{2.3}$$

Poté můžeme definovat samotné PSNR jako:

$$PSNR = 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) = 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right), \tag{2.4}$$

kde MAX_I je maximální možná hodnota pixelu v obrázku (při 8 bitové hloubce $MAX_I = 255$). V rámci RGB modelu je výsledná suma MSE přes všechny složky vydělena třemi. Výsledek nám značí to, že čím vyšší hodnota nám vyjde, tím je obrázek kvalitnější, typickou hodnotou pro komprimované obrazy bývá mezi 30 až 40 dB. Pokud máme totožné obrázky, tak výpočet selhává, protože střední kvadratická chyba MSE je nulová a tudíž PSNR není definované.

2.3.3 SSIM (structural similarity) index

Podobně jako PSNR je SSIM index(structural similarity index) [18] metoda pro měření podobnosti mezi dvěma obrázky.SSIM index byl navržen na základech PSNR a MSE a poprvé vůbec bere v potaz skutečnost, že lidské vnímání je vysoce přizpůsobeno k extrakci strukturální informace zobrazené scény. Hodnoty indexu se pohybují mezi -1 a 1, kde právě hodnota 1 znamená shodný obraz. Tento index se běžně počítá jen na jasové složce Y, kvůli její úzké spojitosti s citlivosti lidského

zraku a tudíž objektivní váze tohoto způsobu hodnocení kvality zobrazeného obrazu. Na rozdíl od metody PSNR, index SSIM respektuje psychovizuální model lidského vidění a poprvé subjektivně hodnotí obraz jako celek. Měřítkem právě mezi dvěma obrazy x a y o rozměrech $N \times N$ je SSIM index definován jako:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}, \tag{2.5}$$

kde $C_i = (K_i L)^2$, L vyjadřuje dynamický rozsah(dynamic range) hodnot pixelů (při 8 bitové hloubce $L = 255$) a $K \ll 1$ jsou malé konstanty. Běžně se volí $K_1 = 0{,}01$ a $K_2 = 0{,}03$. Dále tu máme μ_x a μ_y, které značí vážený průměr obrazu x a y. Pak σ_x^2, σ_y^2 váženou varianci a σ_{xy} kovarianci.

3 VLNKOVÁ TRANSFORMACE

Vlnkovou transformaci (Wavelet transform) signálu je integrální transformace, která se snaží rozlišit jednotlivé části ze kterých je signál složen a nakonec tyto části dobře zobrazit. Tato transformační metoda je algoritmus založený na komparaci analyzovaného signálu s krátkým průběhem, který nazýváme tzv. *vlnkou*. Celá tato operace je přímo založená na této vlnce, kdy se hledá podobnost s originálním signálem za různého roztažení této vlnky nebo její posunutí. Toto celé je výsledek samotné Waveletové transformace.

Užití vlnkové transformace se naskýtá v odstranění různých rušivých faktorů transformovaného signálu a nebo přímo v kompresi, kde dostala dominantní postavení. Pokud signál rozložíme touto transformací, pak můžeme oddělit složky s malým významem a složením získat signál i s mnohonásobně menším obsahem (množstvím dat). Proto se její využití naskýtá v odšumování nebo kompresi obrazových i zvukových signálů a to i v reálném čase.

Základním principem tohoto výpočetního algoritmu, v našem případě zatím spojité transformace CWT (Continuous Wavelet Transformation) je porovnávání zvoleného signálu $x(t)$ se vhodně zvolenou vlnkou (waveletem). Hodnota CWT(τ, s) je výsledkem tohoto porovnávání a udává stupeň podobnosti charakteru vlnky se zvoleným signálem.

Samotná CWT lze pak vyjádřit vztahem

$$\mathrm{CWT}\,(\tau, s) = \frac{1}{\sqrt{|s|}} \int x(t)\psi\left(\frac{t-\tau}{s}\right) \mathrm{d}t, \tag{3.1}$$

kde τ představuje výše zmíněné posunutí a s pak změnu měnu měřítka.

Tento vztah dokazuje že vlnková transformace umožňuje s vysokou přesností lokalizovat průběh signálu ve frekvenční i časovém měřítku.

3.1 Diskrétní vlnková transformace

S příchodem digitalizace signálu, přišli transformace, které dostali svou diskrétní formu v našem případě je to diskrétní vlnková transformace (discrete wavelet transform – DWT) [12]. Tato transformace se dá vyložit jako speciálně vzorkovanou transformaci CWT, která pro pro svůj výpočet využívá rychlý algoritmus jehož součástí jsou filtry typu FIR a následný proces podvzorkování(decimace).

Jelikož použití běžné spojité vlnkové transformace by nebylo příliš vhodné s důvodu obrovského množství dat a nemožné praktické realizace v digitálních signálech, využíváme v praxi právě její diskrétní formu DWT. Právě DWT by se dala také popsat jako diskrétní vlnková transformace s diskrétní časem (DTWT), jelikož

DWT je v základu definována pro spojitá data. Pro přehlednost této práce zkratku DWT budeme právě myslet formu s diskrétním časem. Právě ta si vybere dyadické translace(každé dílčí pásmo je polovina pásma předchozího) a měřítka z celého spektra transformovaných informací.

Vyjádření lze realizovat pomocí ortogonální matice $\mathbf{W}$ o řádu n. A pokud je $\mathbf{y} = (y_1 \ldots, y_n)^{\mathrm{T}}$ vektor délky n, pak jeho vlnkovou transformací je vektor $\mathbf{d} = (d_1 \ldots, d_n)^{\mathrm{T}}$, který lze vyjádřit jako

$$\mathbf{d} = \mathbf{W}\mathbf{y}. \tag{3.2}$$

Poté můžeme díky ortogonalitě $\mathbf{W}$ definovat inverzní vlnkovou transformaci jako

$$\mathbf{y} = \mathbf{W}^{-1}\mathbf{d} = \mathbf{W}^{\mathrm{T}}\mathbf{d}. \tag{3.3}$$

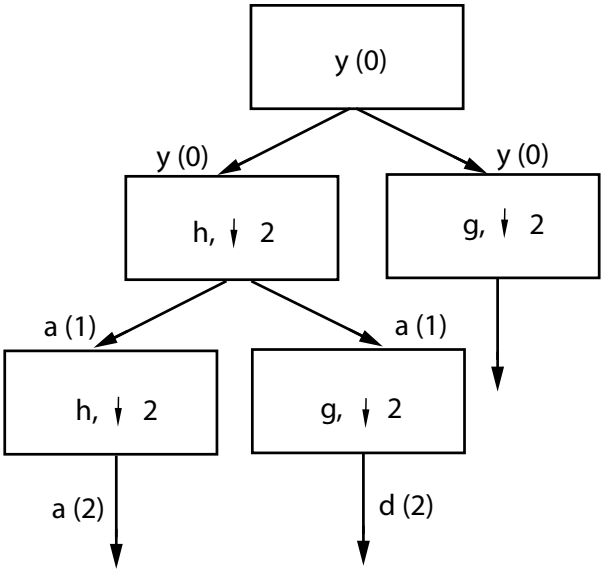
Z výše uvedeného popisu plyne významná vlastnost vlnkové transformace a to její *linearita*.

Realizaci DWT lze provést za pomocí bank filtrů horních a dolních propustí a Mallatova pyramidového algoritmu [12]. Součin vektoru $\mathbf{y}$ a matice $\mathbf{W}$ lze nahradit Mallatovým pyramidovým algoritmem. V tomto případě matici $\mathbf{W}$ nahrazuje filtr typu dolní propust $\mathbf{g}$ a filtr typu horní propust $\mathbf{h}$. Poté vstupní signál $\mathbf{y}$ projde těmito filtry a výstupní signály dále decimujeme (vypustíme každý druhý vzorek signálu). Toto se dá nazvat jako první krok dekompozičního algoritmu. Počet takovýchto kroků označujeme jako stupeň dekompozice d. Dále výstup filtru $\mathbf{g}$ nazýváme detailní koeficienty a výstup filtru $\mathbf{h}$ pak koeficienty aproximační a. Pokud rozklad pokračuje dále, tak získaný vektor za filtrem $\mathbf{g}$ a dále decimovaný je zároveň vstupním vektorem dalšího kroku algoritmu. Krok dekompozičního algoritmu se nazývá hloubka dekompozice. Na obr. 3.1 je znázorněn Mallatův pyramidový algoritmus waveletové dekompozice o hloubce $d = 2$ Je nutno poznamenat, že hloubka dekompozice není libovolná a musí splňovat vztah:

$$d \leq \log_2 s, \tag{3.4}$$

kde s je délka vstupního vektoru.

V opačném případě lze popsat také inverzní algoritmus, který se nazýváme waveletová rekonstrukce. V první řadě rekonstrukce nejdříve nadvzorkujeme vektor aproximačních popřípadě detailních koeficientů tím způsobem, že vložíme nulu za každý prvek vektoru. V druhé řadě následuje filtrace za pomocí filtrů $\tilde{\mathbf{g}}$ a $\tilde{\mathbf{h}}$, které jsou inverzní k filtrům $\mathbf{g}$ a $\mathbf{h}$. Tyto vektory sečteme a výsledkem je první krok rekonstrukce. Tento algoritmus se provádí opakovaně do té doby dokud se nedostaneme na úroveň původního signálu. Aby se předešlo vysokým paměťovým nárokům tak se v praxi používá násobení maticí na rozdíl od DWT.



Obr. 3.1: Mallatův pyramidový algoritmus waveletové dekompozice o 2 stupních.

3.2 Diskrétní vlnková transformace v 2D

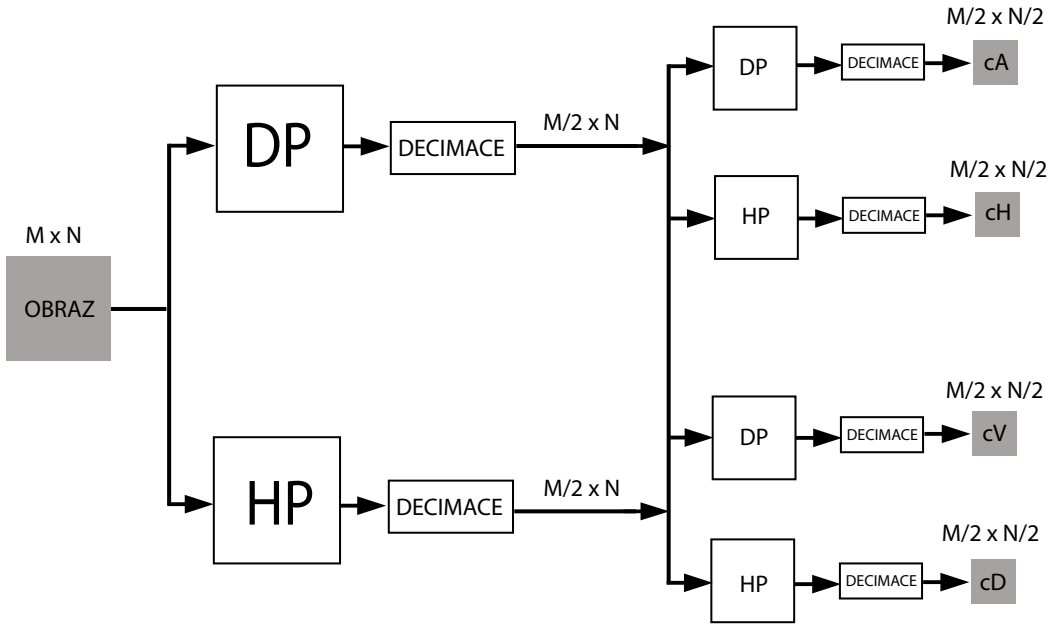
Jelikož při zpracovávání obrazu pracujeme z dvourozměrným signálem, tak v tomto případě využíváme dvourozměrnou vlnkovou transformaci označovanou zkráceně jako 2D DWT [5].

Z vlastnosti vlnkové transformace vyplývá že je separabilní, proto lze zpracovávat nejdříve sloupce a až poté řádky. U 2D DWT tímto způsobem vznikají čtyři druhy koeficientů. (cA – aproximační, cH – horizontální, cV – vertikální, cD – diagonální). Právě aproximační koeficienty cA nesou celkovou informaci o výsledném obrazu po dekompozici. Potom detailní koeficienty cH,cV a cD reprezentují detailní informace v příslušném směru.

Postup lze popsat několika hlavními kroky. V prvním kroku filtrujeme obraz po řádcích a na výstupu se provede decimace poté filturujeme obraz po sloupcích a opět decimujeme. Tímto krokem jsme získali čtyři výsledné obrazy, které jsou co do rozměrů poloviční oproti původnímu vstupnímu obrazu. Další stupeň dekompozice by se aplikoval už jen na detailní koeficienty cA. Celý proces 2D DWT je naznačen na obr. 3.2 kde vidíme rozklad obrazu jednoho stupně dekompozice. Označení HP značí intuitivně horní propust a DP pak dolní propust.

Pokud bychom chtěli rekonstrukci 2D DWT, tak analogickým způsobem jako u 1D DWT vložíme nuly mezi každý prvek neboli nadvzorkujeme.

Abychom celý proces urychlili, tak se používají různé pomocné algoritmy. Mezi nimi lze jmenovat *Lifting Wavelet Transform*, který se používá právě ve obrazovém formátu JPEG2000. Funguje na principu rozdělení původních filtrů na více menších

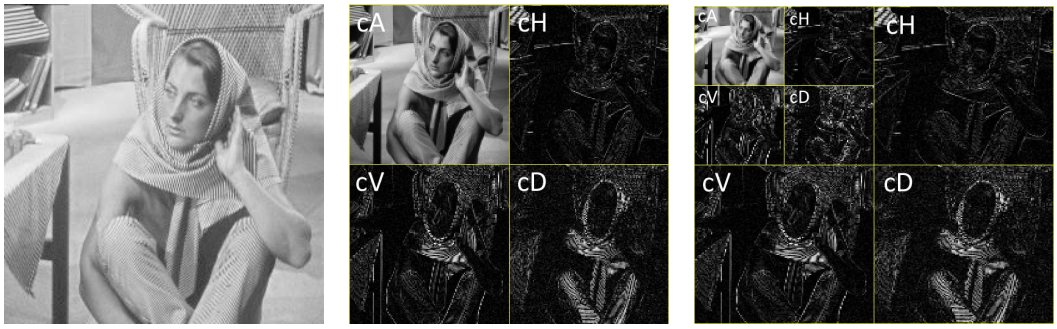


Obr. 3.2: Princip rozkladu obrazu jednoho stupně dekompozice pomocí 2D DWT.

úseků. Tímto dosáhneme snížení výpočetní náročnosti a nároky na paměť.

3.2.1 Dekompozice obrazu

Za použití výše zmíněných postupů si nyní popíšeme a předvedeme dekompozici obrazu [9]. Mějme na vstupu obraz 16×16. Tento obraz představuje matice o stejném rozměru 16×16, kde každý prvek této matice představuje hodnotu jednoho obrazového budu našeho obrazu. Celá matice představuje obraz na, který aplikujeme 2D DWT. Jak je už známi tak nejdříve v rámci řádku a poté v rámci sloupců. Jinými slovy se obraz rozloží na posloupnost po sobě jdoucích řádků a sloupců. Použijeme DWT na řádky a výsledkem je matice o velikosti 16×16, kde levá polovina představuje aproximaci originálního obrazu a ta levá pak detailní informace v podobě vysokých frekvenčních složek (ostré hrany), které jsou odstraněny. Po použití DWT po sloupcích je výsledkem opět matice velikosti 16×16, která v sobě ukrývá nám již známé oblasti koeficientů viz obr. 3.3 cA, cH, cV a cD. Na obrázcích 3.3b a 3.3c lze názorně vidět ořezané vyšší frekvence v jednotlivých hloubkách dekompozice. Vyšší hloubky dekompozice obrazu jsou již analogickou záležitostí.



(a) Původní obrázek (b) 1.stupeň (c) 2.stupeň

Obr. 3.3: Rozklad obrazu při 2 stupních dekompozice.

3.3 Typy vlnek

Hlavním znakem vlnkové transformace je výše zmíněná vlnka, která existuje ve více než 400 různých podobách. Každá vlnka (wavelet) je vhodná pro jiný účel a pro různé úlohy. Tyto vlnky můžeme rozdělit do dvou velkých skupin a to jsou vlnky ortogonální a biortogonální. Mezi typické ortogonální vlnky lze zařadit například rodinu vlnek Daubechies. Důležitá vlastnost ortogonálních vlnek je, že všechny jejich filtry jsou stejně dlouhé a počet prvků je sudý.

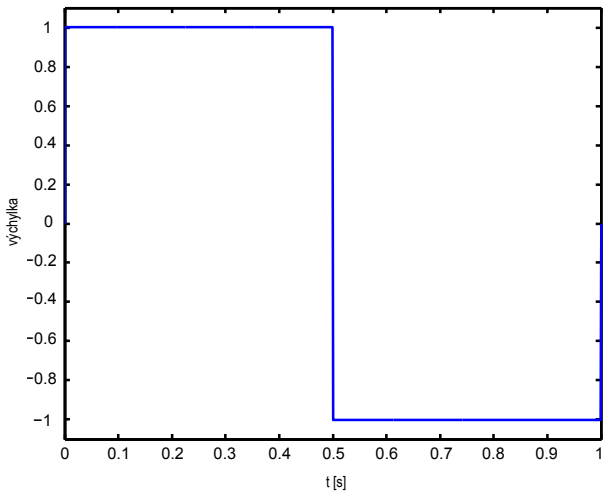
Na druhou stranu vlnky biortogonální se stávají více univerzální a a jejich volnost při návrhu filtrů je velkou výhodou oproti běžným ortogonálním vlnkám. Pokud vezmeme tyto vlnky z pohledu filtrace, tak získáváme čtyři filtry, které nejsou závislé na délce a ani nemusí mít sudý počet prvků. A přesto všechno je stále zaručena podmínka perfektní rekonstrukce.

3.3.1 Haarova vlnka

Mezi základní vlnky lze právě považovat tzv. Haarovou vlnku [14], která je vlastně vůbec první svého druhu. Její počátky sahají do roku 1909 kdy maďarský matematik Alfréd Haar objevil ortonormální systém.

Haarova vlnka je zobrazena na obr. 3.4 kde je dobře zřetelné že její jednoduchý tvar připomíná skokovou funkci. Dále nelze s ní realizovat hladkou rekonstrukci signálu. Výhodou je její rychlý výpočet, ale naopak její nevýhodou je nespojitost funkce. Důležitým poznatkem je že Haarova vlnka je v podstatě vlnkou prvního řádu vlnky zvané Daubechies.

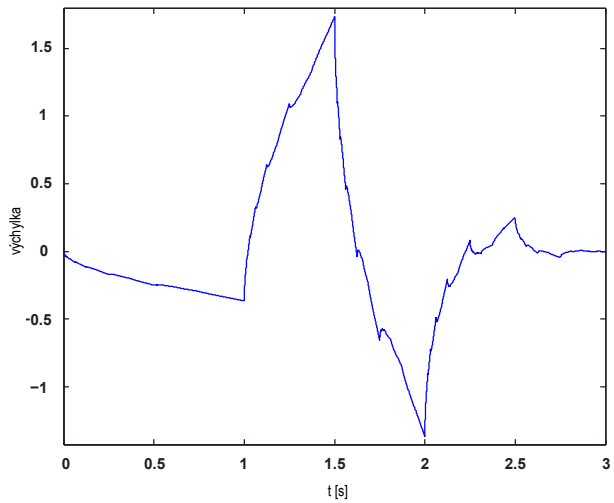
Mezi další důležité vlastnosti této vlnky patří antisymetričnost a délka filtru je $N = 2$.



Obr. 3.4: Haarova vlnka.

3.3.2 Vlnka Daubechies

Další z řad ortogonálních vlnek je již dříve zmíněná samotná rodina vlnek Daubechies [14]. Jejich název je odvozen podle objevilelky, belgické fyzičky a matematičky Ingrid Daubechies. Charakterisické pro tuto rodinu vlnek je že nemají explicitní vyjádření vlnkové funkce a jejich konstrukce co do složitosti je vysoká. Tyto vlnky představují vlnky různých řádů o $N \geq 1$, kde za N v tomto případě můžeme považovat vlnku typu Haar. Vlnku o řádu $N = 2$ můžeme vidět na obr. 3.5.



Obr. 3.5: Vlnka Daubechies o řádu 2.

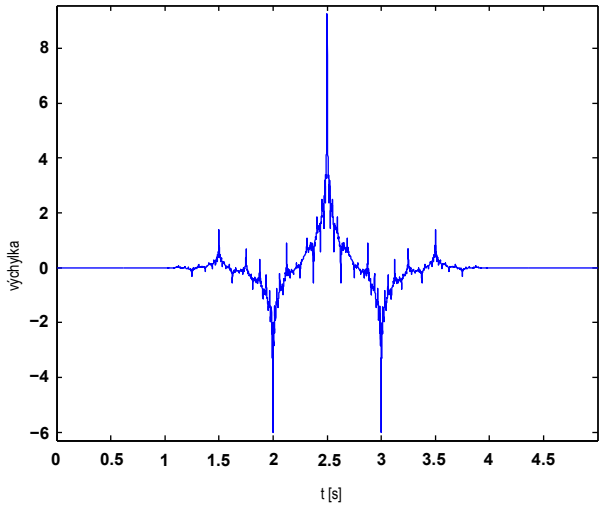
Zajímavostí vlnek Daubechies, že mají známý počet nulových momentů. Jejich konstrukce je realizovaná tím způsobem, že na dané délce nosiče $N - 1$, mají maximální počet momentů, které jsou nulové. Díky tomuto je právě tato vlnka orto-

gonální. Mezi další vlastnosti této vlnky lze zařadit asymetričnost, její kompaktní nosič délky je $2p-1$ (p – počet momentů) a na rozdíl od vlnky Haar je spojitá. Dále je vhodná jak pro použití v CWT tak i DWT.

3.3.3 Biortogonální vlnky

Rodina biortogonálních vlnek také nazývána Cohen-Daubechies-Feauveau wavelet [17](CDF) byla poprvé odvozena v roce 1992 A. Cohenem, již známou I. Daubechiesovou a J. C. Feauveauem. Jedná se o rodinu symetrických vlnek, které nemají explicitní vyjádření. I když jejich název nese jméno Daubechies, tak se nejedná o stejné wavelety. Neboť si nejsou podobné tvarem ani vlastnostmi.

Jejich hlavní využití je právě v kompresi obrazu a to především ve formátu, nám již dobře známým, JPEG2000. Jak je známo z předchozího textu právě formát JPEG2000 podporuje jak ztrátovou tak bezeztrátovou kompresi. Právě pro každý typ této komprese se v tomto formátu používá jiný druh biortogonální CDF vlnky. U bezeztrátové komprese se jedná o biortogonální vlnku CDF 5/3 zvanou také *LeGall 5/3 wavelet* a naopak u ztrátové komprese CDF 9/7 wavelet. Číselná označení (5/3 a 9/7) nám udávají délku použitých filtrů typu dolní propustu. Na obr. 3.6 můžeme vidět biortogonální vlnku CDF 5/3.



Obr. 3.6: Cohen-Daubechies-Feauveau wavelet 5/3 používaný v JPEG2000.

Tyto vlnky se dále používají u video standartu Dirac nebo u otisků prstů *Fingerprint Image Compression Standard* (FBI).

4 OBRAZOVÉ KODEKY ZALOŽENÉ NA VLNKOVÉ TRANSFORMACI

V této části kapitoly se podíváme blíže na algoritmy, které využívají v praxi vlnkovou analýzu obrazu. Mezi ně lze zařadit právě obrazové kodeky EZW, SPIHT a EBCOT. Tyto algoritmy pravě skvěle nabízí jak transformovaná data pomocí DWT efektivně ukládat a hledají optimální poměr mezi kvalitou a výsledným datovým tokem. To všechno mnohem lépe, než při užití klasických kvantovacích metod.

4.1 Embedded Zerotrees of Wavelet transforms

Algoritmus nazývaný jako *Embedded Zerotrees of Wavelet transforms* se zkráceně označuje jako EZW [16]. Jak už bylo zmíněno a z názvu vyplývá, tento algoritmus patří do obrazových kodeků využívajících DWT. Lépe řečeno v případě obrazu 2D DWT. Totiž samotná aplikace 2D DWT a následná komprese z pomocí aritmetického kódování v důsledku postrádá efektivnost. Tudíž je příhodné kódování za pomocí EZW do jednoho proudu dat. Až po aplikaci EZW je tento proud podroben aritmetickému kódování. Zajímavou vlastnosti EZW je fakt, že s každým přidaným bitem, který se přidá do datového toku nám zároveň narůstá přesnost dekódovaného výsledného obrazu. Navíc tento proud není podmíněn délkou a může být tedy libovolný.

Právě EZW využívá jednu důležitou vlastnost DWT, a to je to, že nám klesá pravděpodobnost výskytu vysokých hodnot koeficientů od levého horního rohu po pravý dolní roh. Dále z podstaty DWT vyplývá, že vysoké koeficienty nesou důležitější informace o obraze než ty nízké. Princip EZW tedy spočívá v tom že kóduje koeficienty DWT postupně v několika průchodech. Po každém průběhu je vybírán práh, kterým jsou porovnávány hodnoty koeficientů. V případě pokud je hodnota koeficientu větší než udávaný práh, pak tento koeficient je podroben kódování. Práh je při každém průchodu snižován a ke kódování využívá právě jeden bit, který rozhoduje pouze zda koeficient je vyšší než hodnota uložená v prahu a nebo zda je nižší. Z tohoto vysvětlení plyne, že s každým průchodem se nám do výsledného obrazu přidává více a více informací a obrázek se nám tzv. *postupně doostřuje*. Důležitou podstatou je i to, že čím vyšší je koeficient, tím více bitů je použito k jeho zakódování. Následné bity se zařazují do bitového toku a následná komprese se provede oříznutím tohoto toku na požadovaný počet bitů. Z tohoto lze logicky odůvodnit, že čím více bitů, tím ostřejší hrany obrazu.

4.1.1 Kódování

V úvodu jsme si v základu představili algoritmus EZW. Nyní si pojďme jednotlivé kroky přiblížit více.

Kodér EZW je založený na progresivním kódování (kvalita se postupně zlepšuje) pro kompresi obrazu do bitového toku se zvyšující se přesností. To znamená, že čím více bitů se přidá do bitového toku, tím bude dekódovaný obraz obsahovat více detailů s vlastnostmi podobnými jako u kódování ve formátu JPEG. Toto vyjádření čísla by se dalo přirovnat k číslu π. Každá číslice, kterou přidáme zvyšuje přesnost čísla, ale přes to jej můžeme nastavit na požadovanou přesnost. Progresivní kódování je známé také pod výrazem *embedded encoding*, který vysvětluje E v názvu EZW.

Následuje část Z, která značí *Zerotrees*. Touto částí se budeme podrobněji zabývat v kapitole 4.1.2.

Kódování obrazu pracuje na principu EZW spolu s některými optimalizovanými výsledky v efektivním obrazovém kompresoru a s vlastnostmi takovými, že tok komprimovaných dat může mít jakýkoliv požadovaný bitový tok. Jakýkoliv bitový tok je možný pouze v případě, pokud je informace někde ztracena, potom je kompresor ztrátový. Nicméně, bezztrátová komprese je možná i s kodérem EZW, ale s méně uspokojujícími výsledky.

4.1.2 The Zerotree (Nulový strom)

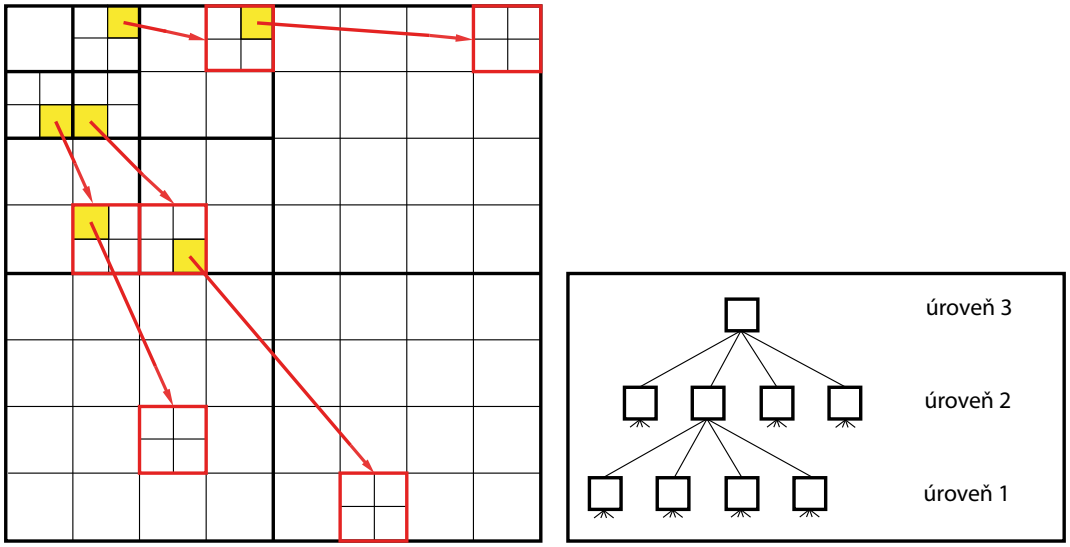
Samotný kodér EZW pracuje s tzv. *The Zerotree* a je založen na dvou důležitých poznatcích:

1. Obrazy v přírodě mají obecně spektrum dolní propusti. Pokud je obraz waveletově transformován, tak energie v subpásmech klesá s měřítkem poklesu (malé měřítko znamená vysoké rozlišení), a proto waveletové koeficienty budou v průměru menší ve vyšších subpásmech než subpásmech nižších. Toto ukazuje, že progresivní kódování je velmi přirozenou volbou pro kompresi waveletově transformovaných obrazů, neboť vyšší subpásma přidávají pouze detaily výsledného obrazu a nikoliv tolik důležité části.
2. Vyšší waveletové koeficienty jsou mnohem důležitější, než nízké waveletové koeficienty.

Tyto dva poznatky jsou využity ke kódování waveletových koeficientů a to postupně v několika průchodech. Pro každý průchod je vybrán práh, vůči kterému jsou všechny waveletovy koeficienty porovnávány. Pokud je waveletový koeficient větší než práh, tak je koeficient zakódován a odebrán z kódovaného obrazu. Pokud je koeficient menší než práh, tak se koeficient ponechá pro další průběh. Dále pokud byli všechny waveletovy koeficienty porovnány s prahem, tak se práh sníží a obraz je se prochází

znova. S každým průchodem se přidává více detailů do kódovaného obrazu. Tento celý proces se neustále opakuje dokud se všechny waveletovy koeficienty nezakódují nebo pokud není splněno jiné kritérium (velikost prahu, maximální bitový tok).

Waveletová transformace převádí signál z časové oblasti do oblasti z časovým měřítkem. To znamená, že waveletové koeficienty jsou dvourozměrné. Pokud bychom chtěli komprimovat transformovaný signál, tak nemůžeme kódovat pouze hodnoty koeficientů, ale také jejich pozici v čase. Pokud je signálem obraz, pak pozici v čase lze nejlépe vyjádřit jako pozici v prostoru. Po provedení waveletové transformace můžeme obraz reprezentovat za pomocí stromů (trees) vzhledem k podvzorkování, které se provádí při transformaci. Koeficient v nízkém subpásmu může předpokládat, že má čtyři potomky v dalším vyšším subpásmu (obr. 4.1). Tyto čtyři potomci mají také čtyři potomky v dalším vyšším subpásmu. Těmto druhům stromu se říká tzv. *quad-trees* (každý kořen má čtyři listy).



Obr. 4.1: Vztahy mezi waveletovými koeficienty v jednotlivých dílčích subpásmech jako *quad-trees*

Nyní můžeme definovat samotný *the zerotree* (nulový strom). *Zerotree* je takový *quad-tree*, jehož všechny uzly jsou stejné nebo menší než kořen. Strom je kódován jako jediný symbol a rekonstruován pomocí dekodéru jako *quad-tree* vyplněný nulami. Tuto definici musíme ještě doplnit o to, že kořen musí být menší než hodnota prahu, proti kterému jsou waveletové koeficienty právě poměřovány.

Kodér EZW využívá toho, že *zerotree* je založeno na načítání snižujících se waveletových koeficientů společně s měřítkem. Předpokládá se to, že pokud je kořen menší než hodnota prahu, tak s velmi vysokou pravděpodobností budou všechny

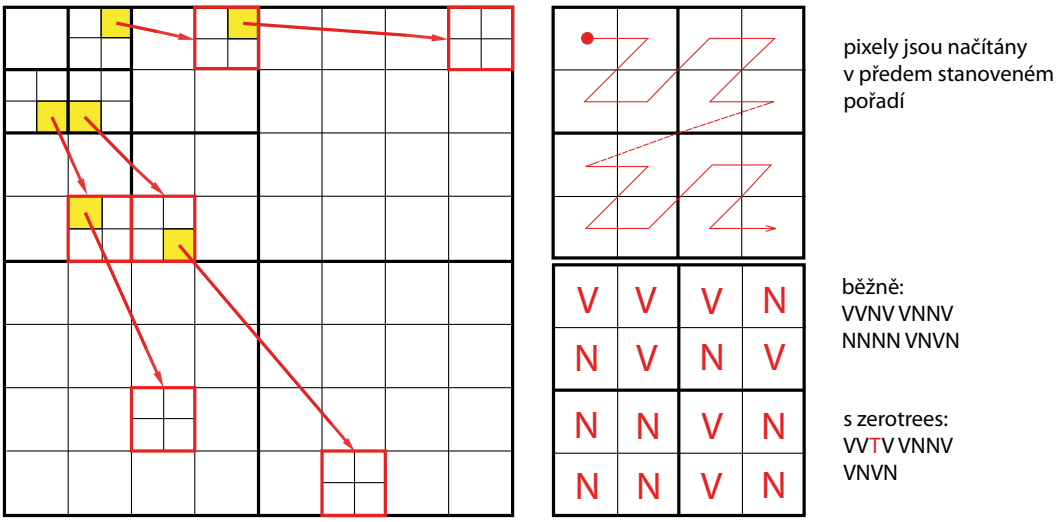
koeficienty v *quad-tree* také menší než hodnota prahu. Pokud se jedná o tento případ, pak celý strom může být kódovaný jako jeden symbol *zerotree*. Poté, když je obraz naskenován v předem stanoveném pořadí (od vysokého měřítka k nízkému), tak je mnoho pozic implicitně zakódováno jako symbol *zerotree*. Samozřejmě toto nebývá vždy pravidlem, ale jak se ukázalo v praxi, tak je pravděpodobnost pravidla *zerotree* obecně velmi vysoká.

4.1.3 Princip kodéru

V předchozích částech jsme si definovali základní poznatky EZW algoritmu. Nyní přistoupíme k samotné kompresi. Přiblížíme si kódování jednotlivých koeficientů a jejich pořadí v jakém budou do kodéru vstupovat.

Pokud bychom vzali nejjednodušší případ, tak bychom přenášeli koeficienty v postupném pořadí (jeden po druhém), ale tento způsob načítání koeficientů není příliš efektivní. V tomto případě by se totiž neuvažovalo postupné snižování koeficientů od levého horního rohu po pravý dolní roh a došlo by ke zbytečnému plýtvání jednotlivých bitů. Mnohem lepší způsob je využití prahové hodnoty a signálu pro dekodér, do kterého vstupují pouze údaje zda je koeficient větší než práh nebo zda je menší než práh. Pokud budeme přenášet i hodnotu prahu, tak dokážeme rekonstruovat opravdu efektivně. Aby došlo k perfektní rekonstrukci je snížena prahová hodnota a celý proces se opakuje, až do té doby kdy bude hodnota prahu menší, než nejmenší hodnota waveletového koeficientu, který chceme přenést. Tento proces můžeme udělat ještě mnohem efektivnější a to odečtením prahu od hodnot větších, než je tento práh. To vede k bitovému toku se vzrůstající přesností, který může být perfektně rekonstruován v dekodéru. Pokud budeme používat předem stanovené pořadí jednotlivých prahových hodnot (např. postupné půlení prahu), pak nemusíme tento práh předávat dekodéru a ušetříme tím určitou šířku pásma. V případě, že je v předem určeném pořadí sekvence dvou působností, tak mluvíme o tzv. *bitplane coding* (kódování bitových hladin), protože prahové hodnoty v tomto případě odpovídají bitům v binární reprezentaci koeficientů.

Jednu podstatnou věc stálé musíme objasnit, a to přenos pozice jednotlivých koeficientů. Jak již bylo zmíněno, tak EZW kódování využívá předdefinované pořadí skenování jednotlivých koeficientů, aby bylo zřejmé v jakém pořadí se budou tyto koeficienty kódovat (obr. 4.2). Díky vlastnostem *zerotrees* je mnoho pozic zakódováno implicitně. Bylo definováno několik skenovacích průchodů, některé z nich můžeme vidět na obr. 4.3. Obecně u nich platí, že jsou nejdříve naskenovány nižší subpásma kompletně, a až pak se přechází k vyšším subpásmům. V počátcích se využíval raster scan (rastrové snímání), ale později bylo definováno několik dalších skenovacích průchodů (např. *Morton scan*). Jak je vidět, tak výběr skenovacího průchodu má



Obr. 4.2: Vztahy mezi waveletovými koeficienty v jednotlivých dílčích subpásmech (vlevo), pořadí skenování (nahoře vpravo) a výsledek *zerotree* (dole vpravo), jako symbol T v kódovacím procesu. Znak V značí koeficienty vyšší než stanovená hodnota prahu a znak N pak nižší než práh. *Zerotree* symbol T nahradil čtyři symboly N v levé dolní části a N v levé horní části.

vliv na výsledek komprese.

4.1.4 Algoritmus

V předchozích částech jsme si představili schéma kódování koeficientů a jejich pořadí kódování. Nyní můžeme přistoupit přímo k algoritmu.

Výstupní datový tok EZW, by měl začínat s informacemi potřebnými pro synchronizaci dekodéru. Mezi minimální požadované informace pro dekodér patří úroveň použité waveletové dekompozice a počáteční práh (za předpokladu že bude použita pro rekonstrukci také waveletová transformace). Dodatečně mohou být poslány rozměry obrazu a hodnoty obrazu. Poslání hodnot obrazu je užitečné v případě, že jsou odstraněny z obrazu před kódováním. Po nedokonalé rekonstrukci dekodér může nahradit nedokonalé hodnoty původními.

Prvním krokem EZW kódovacího algoritmu je stanovení počátečního prahu. Pokud budeme uvažovat kódování bitových hladin, pak počáteční práh t_0 bude

$$t_0 = 2^{FLOOR(\log_2(MAX(|\gamma(x,y)|)))}, \tag{4.1}$$

kde MAX znamená maximální hodnota koeficientu a $\gamma(x, y)$ pak koeficient na souřadnicích x, y.

Dalším krokem jsou dva typy průchodů, které se využívají ke kódování. Prvním typem průchodu je tzv. *the significance pass* nebo také *dominant pass*. V českém

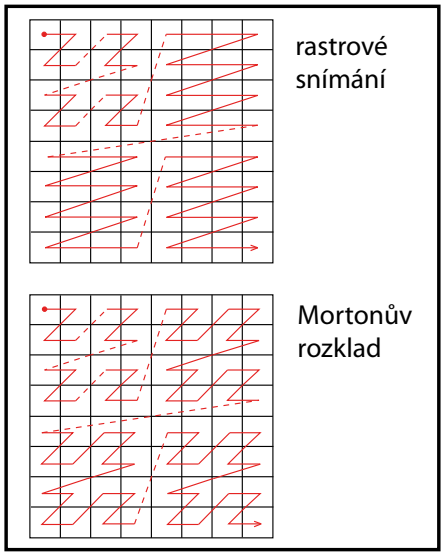
ekvivalentu pak dominantní průběh. Průchod skenuje každý koeficient podle stanoveného pořadí a zapíše na výstup jeden ze čtyř znaků podle typu koeficientu:

1. POS = Significant Positive (**P**) - Symbol je použit pro případ, kdy koeficient je větší než „práh“.
2. NEG = Significant Negative (**N**) - Symbol je použit pro případ, kdy koeficient je větší než „−práh“.
3. ZTR = Zero Tree Root (**T**) - Symbol značí výskyt *zerotree* (nulového stromu).
4. IZ = Isolated Zero (**Z**) - Symbol je použit pokud je koeficient menší než „práh“, ale není kořenem *zerotree*.

Všimněme si, že pokud bychom chtěli určit zda je koeficient kořen *zerotree* nebo *isolated zero* museli bychom skenovat celý *quad-tree*. Tímto je jasné, že toto určování by zabralo nějaký čas. Zároveň se musí sledovat to, aby se na výstup nedostali znova hodnoty, které již byli označeny jako *zerotree*. Proto se musí uvažovat paměť, která si tyto hodnoty bude pamatovat.

Nakonec jsou všechny koeficienty, které jsou v absolutní hodnotě větší než hodnota aktuálního prahu extrahovány a uloženy bez jejich znaménka do tzv. *subordinate* seznamu a jejich pozice jsou vyplněny nulami. Tímto se zabrání, aby se koeficienty nekódovali znovu.

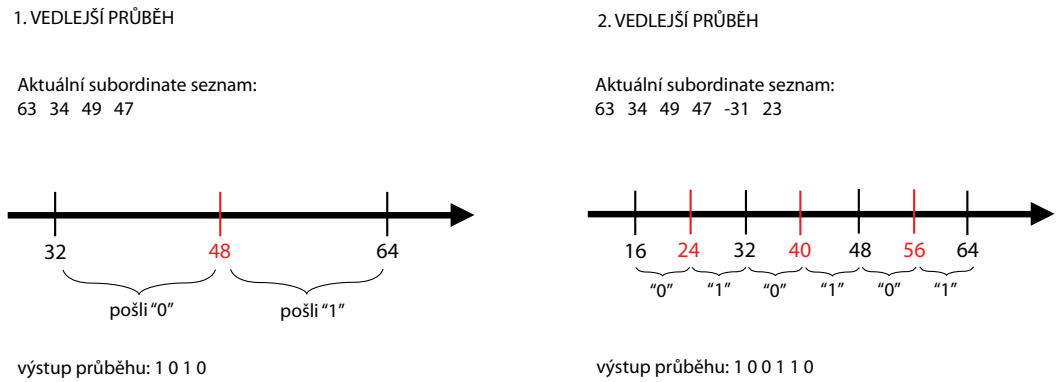
63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4



Obr. 4.3: Ukázka hodnot waveletových koeficientů (vpravo) a ukázka dvou typů průchodů (vlevo). V naší ukázce je použít *Mortonův rozklad*.

Druhým průchodem je tzv. *the refinement pass* nebo také *subordinate pass*. Jako český ekvivalent pak jako vedlejší průběh. Tento průběh následuje hned po dominantním a pracuje pouze s koeficienty, které byli uloženy do *subordinate* seznamu.

Aktuální hodnota prahu se rozdělí na polovinu ($\frac{prah}{2}$). Jednotlivým koeficientům přiřadí hodnotu 0 nebo 1. Hodnotu 0 pokud je koeficient menší než polovina aktuálního prahu ($\frac{prah}{2}$) a hodnotu 1 koeficientu menší než polovina aktuálního prahu ($\frac{prah}{2}$). Výstupní tok vedlejšího průběhu má stejný počet 0 a 1, jako má dominantní průběh symbolů **P** a **N** dohromady. Tento průchod je více přiblížený na obr. 4.4. Jsou zde zobrazeny aktuální hodnoty *subordinate* seznamu pro první dva průchody a jejich jednotlivé intervaly pro přehlednost (kdy je na výstup zapsána hodnota 0 a kdy 1). Je vidět, že s každým průchodem se polovina aktuálního prahu ($\frac{prah}{2}$) zmenšuje (interval se zužuje), a tudíž rekonstruované koeficienty jsou přesnější. Ostatní průchody, by pokračovali obdobným způsobem.



Obr. 4.4: První dva vedlejší průchody a vysvětlení jejich významu.

Po druhém průchodu je hodnota prahu snížena a celý proces kodéru se opakuje. V případě dekodéru se jedná pouze o inverzní záležitost. Nakonec je tok symbolů podroben vhodným aritmetickým kódováním.

4.1.5 Příklad

Nyní již víme jak EZW funguje. Pro lepší pochopení uvedeme konkrétní příklad, který lze vidět na obr. 4.3. Označení `D` ve výpisu vyjadřuje dominantní průchod a znak `V` pak vedlejší průchod. Číselné označení vyjadřuje pořadí jednotlivých průchodů.

```
D1: pnztpttttztttttttptt
V1: 1010
D2: ztnpttttttt
V2: 100110
D3: zzzzzppnppnttnnptpttnttttttttptttptttttttttptttttttttttt
V3: 10011101111011011000
```

```
D4: zzzzzzztztznzzzzpttptpptpnptnttttttptpnpppptttttptptttpnp
V4: 1101111101100100000111011010000100101011100
D5: zzzzztzzzzztpzzzttptttttnptpptttptttnppntttttpnnptttptpptttt
V5: 10111100110100010111110101101100100000000011011011001100011l
D6: zzzttztttztttttnnttt
```

Vedlejší průchod na poslední úrovni je možno vynechat, protože polovina aktuálního prahu ($\frac{prah}{2}$) je v tomto okamžiku již nulová. Z toho je zřejmé, že jej vykonávat nemá smysl.

4.2 Set Partitioning in Hierarchical Trees

Dalším z řad algoritmů využívaných při kompresi obrazů a za pomocí DWT je algoritmus nazývaný *Set Partitioning in Hierarchical Trees* [11], který se zkráceně označuje jako SPIHT. Stejně jako EZW, tak i SPIHT pracuje s koeficienty vzniklé pyramidovým rozkladem 2D DWT. Algoritmus vychází z výše zmíněného algoritmu EZW a v mnoha ohledech ho zdokonaluje. Algoritmus pracuje z celočíselnými koeficienty vzniklé z dekompozice obrazu. Princip spočívá v tom, že na výstupu kodéru je posloupnost bitů, jejíž délka určuje výslednou kvalitu rekonstruovaného obrazu. Z toho plyne, že u SPIHT je možné kompresní postup kdykoliv zastavit. Pokud tento algoritmus necháme proběhnout až do konce, tak obrazová data na výstupu představují téměř bezztrátový obraz ochuzený pouze chybou při zaokrouhlování výpočtů. Z podstaty SPIHT dále plyne, že bitový tok je progresivně dekódovatelný, jinými slovy obrazová kvalita se s každým průchodem zlepšuje.

Pokud si vezmeme samotný algoritmus, tak ten je založen v podstatě na dvou základních principech. Prvním je, že budeme uvažovat pouze jednu bitovou hladinu. Ta je na počátku nastavena na bitovou hladinu nejvyššího bitu n největšího koeficientu. Takovýmto způsobem se vyberou a seskupí pouze takové koeficienty, které jsou buď vetší nebo rovné 2^n. Jelikož jsou očekávány největší koeficienty (koncentrace energie) v případném nejvyšším stupni dekompozice dekompozičního obrazce, tak je tento algoritmus zaměřen právě na toto subpásmo.

Druhým principem je prostorové umístění koeficientů v jednotlivých subpásmech. V každém subpásmu jsou totiž obsaženy koeficienty, které se vztahují k příslušné oblasti v originálním obraze. Toto všechno se dá znázornit za pomocí stromové struktury (viz. kapitola o EZW 4.1), ze které tento kompresní algoritmus čerpá. Namísto jednotlivých koeficientů bereme v potaz celý strom. Na obr. 4.5 můžeme vidět strukturu stromu, kde je vidět, že kořeny se nacházejí v nejvyšším subpásmu. Stejně jako u EZW má každý uzel stromu čtyři přímé potomky.

Pokud vezmeme srovnání SPIHT a EZW, tak algoritmus SPITH je sice co do náročnosti implementace horší, ale při stejné kvalitě se dosahuje kratšího výstupního bitového toku než v případě EZW.

4.2.1 Označení koeficientů a množin koeficientů

V úvodu jsme si základně představili kódovací algoritmus SPIHT. Nyní přikročíme k definování jednotlivých typu koeficientů a množin koeficientů, se kterými se v algoritmu pracuje. Podle [15] se zavedlo toto označení:

- $c_{i,j}$ ... hodnota koeficientu na x, y na souřadnicích i, j
- (i, j) ... prvek (uzel, pixel, koeficient) daný souřadnicemi (i, j)
- $\mathcal{O}(i, j)$ ... množina souřadnic všech přímých potomků uzlu (i, j)
- $\mathcal{D}(i, j)$ ... množina souřadnic všech potomků uzlu (i, j)
- $\mathcal{H}$... množina všech kořenů prostorově orientovaných stromů (koeficienty v nejvyšším stupni dekompozice)
- $\mathcal{L}(i, j) = \mathcal{D}(i, j) - \mathcal{O}(i, j)$

Jednotlivé označení koeficientů je možno nalézt právě na obr. 4.5, kde jsou jednotlivé množiny a prvky dobře viditelné.

Pro každý prvek můžeme definovat prostorovou závislost jednotlivých prvků (i, j) ve stromu. Přesněji závislost přímých potomků prvku (uzlu). Vyjádření by se dalo shrnout takto:

$$\mathcal{O}(i, j) = \{(2i, 2j), (2i, 2j + 1), (2i + 1, 2j), (2i + 1, 2j + 1)\}. \tag{4.2}$$

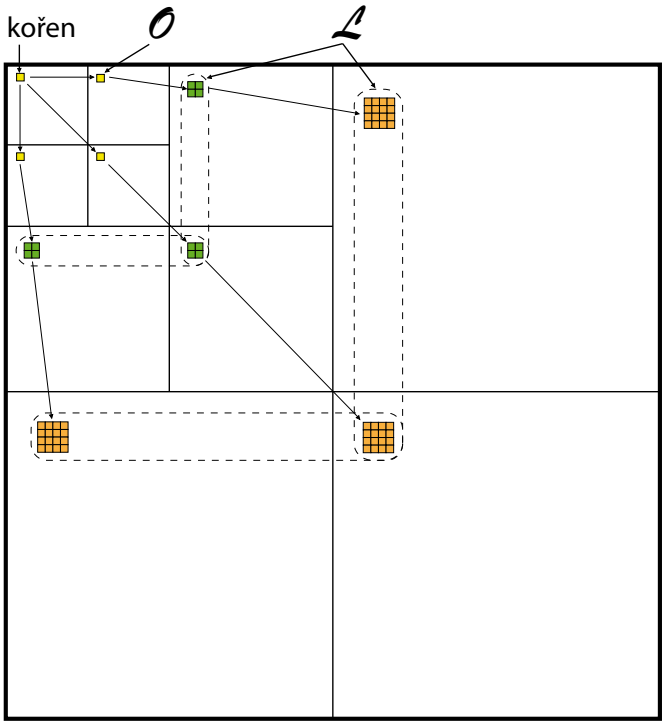
Tento vztah platí pro všechny prvky stromu kromě těch, kteří leží v nejvyšším nebo nejnižším subpásmu.

4.2.2 Funkce významnosti

Dříve jsme zmínili, že algoritmus SPIHT pracuje v postupných krocích a to po bitových hladinách, které se postupně snižují (dekrementují) jako proměnná n. V této souvislosti byla definována tzv. *funkce významnosti* $S_{\mathrm{n}}(\mathcal{T})$ prvků a množin prvků $\mathcal{T}$, která platí pokaždé pro aktuální hladinu n:

$$S_{\mathrm{n}}(\mathcal{T}) = \begin{cases} 1, & \text{pro } \max\{|\, c_{i,j} \,|\} \geq 2^n \qquad \text{kde } (i, j) \in \mathcal{T} \\ 0, & \text{jinak,} \end{cases} \tag{4.3}$$

kde jeden koeficient je obecně $\mathcal{T} = (i, j)$.



Obr. 4.5: Ukázka kořene stromu a jednotlivých definovaných množin v pyramidové dekompozici

Kdybychom vyjádřili tuto funkci slovně, tak se jedná o to zda největší z absolutních hodnot prvků dané množiny je větší nebo rovná hodnotě 2^n, kde n (jak bylo zmíněno) je aktuální bitová hladina. V případě, že je výsledek funkce $S_{\mathrm{n}}(\mathcal{T}) = 1$, pak množina je v daném průběhu významná, v případě $S_{\mathrm{n}}(\mathcal{T}) = 0$, pak logicky nevýznamná. S nevýznamnými množinami se v daném kroku, který určuje hodnota hladiny n, vůbec neuvažuje. Pokud se tato funkce aplikuje na jednoprvkové množiny, tak toto rozhodnutí je konečné a na výstup je poslána konečná hodnota. V případě víceprvkových množin, které byli v daném kroku shledány významnými (stromy) dojde k rozdělení na podmnožiny neboli *podstromy* označované jako $\mathcal{T}_m$. Na tyto podstromy je znova aplikována funkce významnosti. Cílem tohoto procesu je najít nevýznamné podstromy, které by se reprezentovali jediným bitem.

4.2.3 Algoritmus

V předchozích částech jsme si představili základní označení a funkci významnosti. Nyní přistoupíme k vysvětlení funkce kodéru a v další části k tomu, jak se musí uvažovat v případě implementace. Nakonec jak se pracuje v případě dekodéru.

Kodér

Kodér pracuje ve třech základních krocích. Počáteční rozdělení do stromů je pak:

1. vytvoří se $\mathcal{D}(i,j)$ pro každý $(i,j) \in \mathcal{H}$
2. je-li množina $\mathcal{D}(i,j)$ významná ($S_{\mathrm{n}}(\mathcal{D}(i,j))= 1$), pak se rozdělí na $\mathcal{L}(i,j)$ a na čtyři jednoprvkové množiny $(k,l) \in \mathcal{O}(i,j)$ (jinými slovy na čtyři přímé potomky).
3. je-li množina $\mathcal{L}(i,j)$ významná ($S_{\mathrm{n}}(\mathcal{L}(i,j))= 1$), pak se rozdělí na čtyři množiny $\mathcal{D}(k,l)$, kde $(k,l) \in \mathcal{O}(i,j)$ (jinými slovy vzniknou čtyři množiny s přímými potomky jako novými kořeny)

Pokud budeme mluvit o praktické implementaci, tak v tomto případě musíme uvažovat, mírně odlišně. Informace o pořadí testovaných podmnožin se uchována ve třech dynamicky se měnících seznamech.

- **LIS** = *List of insignificant sets* - seznam nevýznamných množin
- **LIP** = *List of insignificant pixels* - seznam nevýznamných prvků (pixelů)
- **LSP** = *List of significant pixels* - seznam významných prvků (pixelů)

Stejně jako v obecném postupu, tak i zde se souřadnice prvků jednoznačně identifikují souřadnicemi (i,j). V případě seznamů LSP a LIP, jsou uloženy hodnoty koeficientů (pixely). V seznamu LIS je každý prvek reprezentován množinou, a to $\mathcal{D}(i,j)$ nebo $\mathcal{L}(i,j)$. Díky tomu ještě rozdělíme tento seznam na dva typy. Typ A v případě, že se bude jednat o množinu $\mathcal{D}(i,j)$ a typ B v případě množiny $\mathcal{L}(i,j)$.

Jak v případě EZW, tak i zde bylo definovány dva různé průběhy. Prvních z nich je *řadící průběh*, kde jsou testovány prvky v LIP (v minulém kroku shledány nevýznamnými) na významnost pomocí funkce významnosti a ty, které jsou významné přesune algoritmus do seznamu LSP. Obdobným způsobem jsou procházeny množiny v seznamu LIS, kde v případě, že jsou významné dojde k jejich rozdělení na nové podmnožiny a původní množina je pak následně ze seznamu LIS odstraněna. Pokud mají množiny více než jeden prvek, pak jsou přidány na konec seznamu LIS, zbývající jednoprvkové množiny pak nakonec seznamu LIP nebo LSP (podle toho, zda jsou v daném kroku významné nebo nevýznamné). Následně jsou prvky LSP dále zpracovány v druhém *upřesňovacím průběhu*.

Popis samotného algoritmu

Nyní se dostáváme k samotnému popisu algoritmu SPIHT, kde názorně předvedeme celý proces. Můžeme jej představit ve čtyřech základních krocích: *Inicializace*, *Řadící průběh*, *Upřesňovací průběh* a *Úprava kvantizačního kroku*.

1. **Inicializace:**
   $n = \left\lfloor \log_2 \left(\max_{(i,j)} \{|c_{i,j}|\} \right) \right\rfloor$, LSP je prázdný, souřadnice prvků $(i,j) \in \mathcal{H}$do seznamu LIP a do seznamu LIS jako typ A pouze ty, co mají nějaké potomky.

2. **Řadící průběh:**
   (a) Pro každý prvek (i,j)v seznamu LIP se provede:
      i. na výstup bit $S_{\mathrm{n}}(i,j)$
      ii. je-li $S_{\mathrm{n}}(i,j) = 1$, přesune se prvek (i,j) do seznamu LSP a na výstup se zapíše znaménkový bit hodnoty koeficientu na souřadnicích (i,j)
   (b) Pro každý prvek (i,j) v seznamu LIS se provede:
      i. Je-li prvek (i,j) typu A, pak se provede:
         A. na výstup bit $S_{\mathrm{n}}(\mathcal{D}(i,j))$
         B. je-li $S_{\mathrm{n}}(\mathcal{D}(i,j)) = 1$, pak se provede:
            - pro každý prvek $(k,l) \in \mathcal{O}(i,j)$ se provede:
              - na výstup bit $S_{\mathrm{n}}(k,l)$
              - je-li $S_{\mathrm{n}}(k,l) = 1$, pak se přidá prvek (k,l) na konec seznamu LSP a na výstup znaménkový bit $c_{i,j}$
              - je-li $S_{\mathrm{n}}(k,l) = 0$, pak se přidá prvek (k,l) na konec seznamu LIP
            - není-li $\mathcal{L}(i,j)$ prázdný, pak se přesune prvek (i,j) na konec seznamu LIS jako typ B, je-li prázdný odstraní se (i,j) ze seznamu LIS
      ii. Je-li prvek (i,j) typu B, pak se provede:
         A. na výstup bit $S_{\mathrm{n}}(\mathcal{L}(i,j))$
         B. je-li $S_{\mathrm{n}}(\mathcal{L}(i,j)) = 1$, pak se provede:
            - přidají se všechny $(k,l) \in(\mathcal{O}(i,j))$ na konec seznamu LIS jako typ A
            - odstraní se (i,j) ze seznamu LIS

3. **Upřesňovací průběh:**
   Na výstup se zapíše n-tý nejvyšší bit pro každý prvek (i,j) v seznamu LSP, mimo těch, které byly přidány v posledním řadícím průběhu.

4. **Úprava kvantizačního kroku:**
   Sníží se n o jeden a pokračuje se znovu řadícím průběhem (krok 2) dokud je splněna podmínka $n \geq 0$. Pokud není tak se proces ukončí.

Ještě je nutno dodat, že pokud mluvíme o kroku 2(b) kde se přidává prvek na konec seznamu je důležité, že se s tímto prvkem musí pracovat ještě před skončením aktuálního řadícího průběhu. Celý tento proces kódování lze kdykoliv zastavit a přes to bude výsledek vždy dekódovatelný. Právě délka výstupního toku bitů nám přímo udává požadovanou kvalitu výsledného obrazu.

Dekodér

V předchozí části jsme si přiblížili dekodér a celý kódovací algoritmus. Součásti algoritmu je i samozřejmě dekodér. Aby dekodér správně fungoval bude potřebovat počáteční hodnotu n (bitovou hladinu), dále rozměry kódovaného obrazu a stupeň použité waveletové dekompozice. Díky těmto údajům může dekodér sledovat podle vstupních dat cestu kodéru. Myšlenku dekódování lze v podstatě uvažovat jako opačný proces ke kódování (místo výstupu budeme uvažovat vstup).

Důležitou funkci dekodéru je rekonstruovat waveletové koeficienty, zapsat je do výsledného obrazu. Toto se provádí až tehdy, když jsou souřadnice koeficientu přesunuty do seznamu LSP. V této chvíli je naprosto jasné, že absolutní hodnota koeficientu $c_{i,j}$ leží na intervalu $\langle 2^n, 2^{n+1})$ a zároveň je také známe znaménko příslušného koeficientu. Hodnota, která se právě rekonstruovala se prozatím nastaví na $\hat{c}_{i,j} = \pm 1{,}5 \cdot 2^n$. Poté v upřesňovacím průběhu se od prvku odečte nebo k němu přičte (dle znaménka) hodnota 2^{n-1}. Z toho vyplývá postupné zpřesňování koeficientu s každým dalším krokem.

Za nezbytnou podmínku je nutno považovat také předpoklad přesné bitové synchronizace dekodéru a kodéru. Kdyby tomu tak nebylo, tak i sebemenší odchylka může způsobit znehodnocení celého dekódovacího procesu.

4.2.4 Barevná informace

Předchozí metoda bylo popsána pouze pro případ, kdy se jedná o obraz ve stupních šedi a uvažuje se pouze jasová složka Y. Barevnou informaci v obrazu tvoří (jak již bylo zmíněno) složky C_B a C_R. Jako první by se dala uvažovat logická úvaha, že bychom kódovali každou barevnou rovinu zvlášť. Avšak tento postup se jeví jako nevýhodný, kvůli problematické synchronizaci kódovacích postů všech rovin a také by se složitě využívala možnost progresivního dekódování. Proto byly navrženy modifikace pro algoritmus SPIHT. Mezi ně lze zařadit *metodu kombinaci jasových složek* a metodu zvanou CSPIHT.

4.3 Embedded Block Coding with Optimal Truncation

Tento kódovací algoritmus je velmi dobře známý z obrazového formátu JPEG2000. Poté co se provede kvantizace tak každé subpásmu je rozděleno do tzv. *packet partitions*. Tímto pojmem jsou chápany oblasti složené s nepřekrývajících se čtverců a jsou běžně voleny tak, aby tvořily bloky v původním obrazu napříč podpásmy. Pokaždé jsou *packet partition* rozděleny do bloků stejných velikostí a poté se vezmou všechny bity všech koeficientů v každém bloku a je na ně aplikován kódovací algoritmus zvaný *Embedded Block Coding with Optimal Truncation*, zkráceně EBCOT [2], [3].

Tento algoritmus funguje na principu kódování, že zakóduje všechny koeficienty v bloku a to od nejvíce významného po nejméně významný. Provede se to tak, že se rozdělí koeficienty do jednotlivých bitových rovin. Pokud bitové roviny obsahují nuly, tak dojde k jejich přeskočení, ale jejich počet se uchová. Samotné kódování započne, až po nalezení první nenulové (obsahuje alespoň jednu jedničku) bitové roviny. Poté se postupně z bitové roviny načtou bity po blocích obsahující čtyři řádky, neboli v bitovém měřítku po čtyřbitových sloupcích. Nakonec je každá rovina kódována třemi průchody, které nesou názvy *Significance Propagation*, *Magnitude Refinement* a *Cleanup Pass*. Prvním bývá vždy *Cleanup Pass*. *Significance Propagation* znaméná to, že se kódují ty bity, které se dají označit jako nedůležité, ale zároveň je známo že sousedí alespoň v jednom z 8 směru s důležitým bitem. *Magnitude Refinement* poté kóduje všechny bity, které leží na místech, kde v minulé rovině leželi důležité bity. Nakonec všechny zbylé bity zakóduje *Cleanup Pass*. Příklad blokového kódování lze vidět na obr. 4.6.

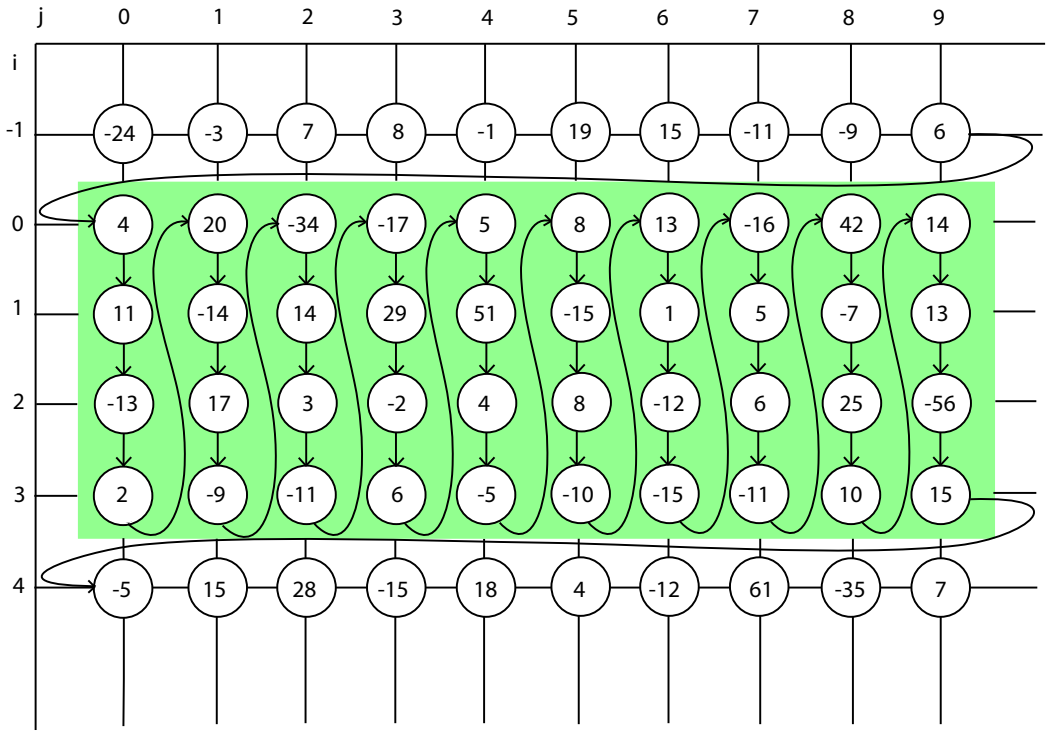
4.3.1 Příklad použití

Pro lepší vysvětlení algoritmu EBCOT zvolíme rovnou formu o ukázky samotného kódování.

V úvodu jsme si představili tři hlavní kódovací průběhy, které se v EBCOT využívají. Dále se v EBCOT využívají čtyři typy základních kódování

- *Run-Length* (**RL**),
- *Zero Coding* (**ZC**),
- *Magnitude Refinement* (**MR**),
- *Sign Coding* (**SC**),

které budeme označovat dále v textu uvedenými značkami.



Obr. 4.6: Ukázka blokového kódování v EBCOT

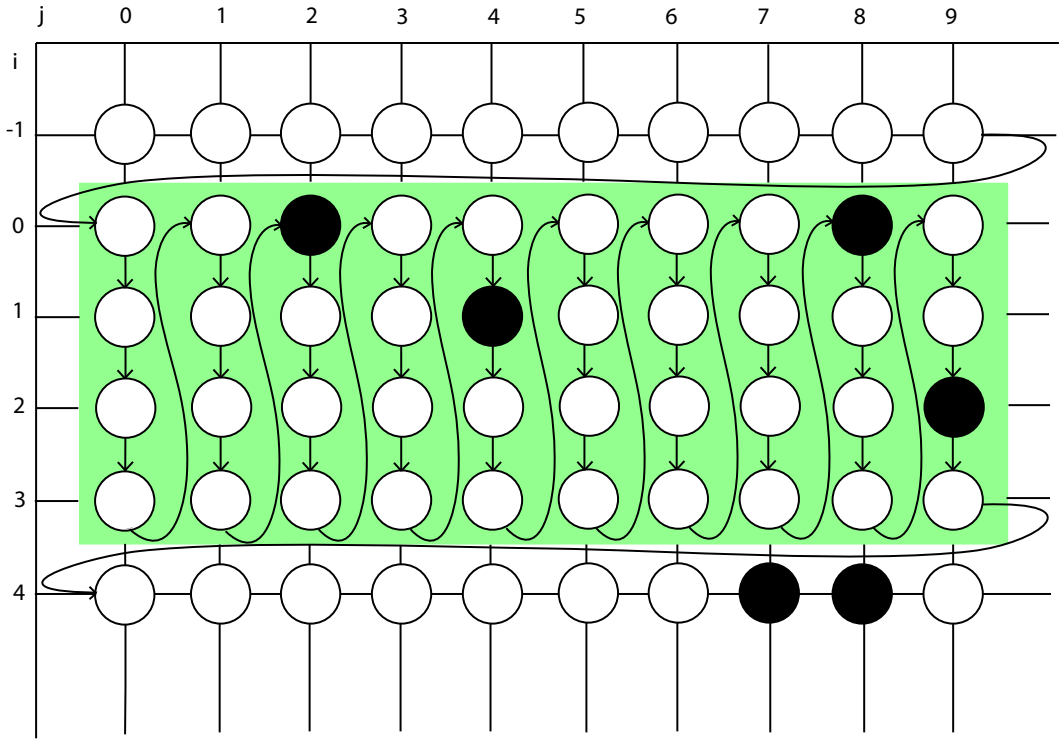
Kodér EBCOT kóduje waveletové koeficienty po blocích, tudíž se řadí mezi blokové kodéry. Obecná velikost těchto bloků je 64×64 nebo 32×32. Část bloku, na kterém budeme kodér vysvětlovat můžeme vidět na obr. 4.6. Tato část se konkrétně nachází v LH (cH) subpásmu. Čísla v jednotlivých buňkách reprezentují waveletové koeficienty po kvantizaci. Tyto koeficienty jsou v rozumném pořadí sloupec po sloupci o čtyřech koeficientech skenovány (načítány). Toto načítání je názorně předvedeno opět na obr. 4.6, kde první řádek koeficientů představuje předchozí blok a poslední řádek pak následující blok.

Nejvyšší hodnota koeficientu je 61 (na pozici $(4, 7)$). V tomto bloku je tedy 6 úrovní bitových hladin ($2^6 = 64$). Ty jsou vyjádřeny od 0 do 5, kde hodnota 5 značí nejvýznamnější bitovou hladinu. Toto číslo bitové hladiny je obsaženo v bitovém toku, aby dekodér věděl, na které bitové hladině započne.

4.3.2 Kódování první bitové hladiny

První bitová hladina ($n = 5$) je zobrazena na obr. 4.7. Na této bitové hladině se využívá pouze *Cleanup Pass* (čistící průběh). Průchod zvaný *Significance Propagation* nemůže být použit, protože se před touto bitovou hladinou nenachází žádný významný waveletový koeficient. Jako nevýznamný waveletový koeficient myslíme ten koeficient, jehož významnost se nemůže šířit kolem významných koeficientů.

Průchod zvaný *Magnitude Refinement* nemůže být použit ze stejných důvodů. Významné koeficienty musí být nalezeny předtím, než se jejich hodnota bude zpřesňovat (měnit). Z toho vyplývá, že jediným kódovacím průběhem v první bitové hladině je *Cleanup Pass*. Tento průchod se nazývá *clenup* (čistící), protože se používá k zakódování všech bitů, které nebyli kódovány v *Significance Propagation* průchodu nebo v průchodu *Magnitude Refinement*.



Obr. 4.7: Bitová hladina na úrovni 5. Černé buňky představují koeficienty, které jsou v této hladině významné.

Průchod Cleanup Pass

Tento kódovací proces začíná ve sloupci $j = 0$. Před kódováním tohoto sloupce víme, že se zde nenacházejí žádné významné waveletové koeficienty. Kromě toho není znám žádný významný koeficient v sousedství tohoto sloupce. Pokud jsou tyto dvě podmínky splněny, pak se kodér přepne do *Run-Length* režimu. Sloupec $j = 0$ je pak následně skenován (načítán). V bitové hladině $n = 5$ se v tomto sloupci nenachází žádný významný koeficient. Z tohoto důvodu je symbol 0 na výstupu dekodéru (RL) indikován jako sloupec nul.

Stejným způsobem se postupuje u nevýznamných koeficientů i v jejich sousedstvích ve se sloupci $j = 1$. Kodér zůstává v režimu *Run-Length*. Jelikož v tomto

sloupci není žádný významný koeficient (v bitové hladině $n = 5$), tak jako v předchozím případě je zapsán na výstup (RL) symbol 0 (jako indikovaný sloupec nul).

V případě kódování sloupce $j = 2$ kodér zůstává v režimu *Run-Length* do té doby, dokud jsou splněny obě podmínky. Avšak koeficient $(0, 2)$ je významný (v bitové hladině $n = 5$). V tom případě je na výstup (RL) poslán symbol 1 a ten indikuje ukončení režimu *Run-Length*. Navíc kodér indikuje pozici prvního významného koeficientu sloupce symbolem 00 (UNI), který pošle na výstup (v UNIFORM - jednotné souvislosti jsou symboly považovány za *equi-probable* neboli ekvivalentně pravděpodobné). Toto umožňuje dekodéru znát přesnou pozici toho kde se *Run-Length* režim ukončí. Znaménko každého waveletového koeficientu je kódováno jakmile je koeficient shledán významným. Proto je znaménko koeficientu $(0, 2)$ kódováno nyní.

Pro kódování znaménka se v algoritmu EBCOT použita predikce (předpověď) založená na znaménkách čtyř sousedů. Tyto predikce jsou uvedeny v tab. 4.1 a to pro subpásma LL (cA), LH (cH) a HH (cD). Hodnota χ^{-h} udává znaménka dvou horizontálních sousedů a hodnota χ^{-v} pak znaménka dvou vertikálních sousedů. V případě subpásma HL (cV) jsou v tabulce tyto veličiny inverzní. Hodnota $\bar{\chi}$ se rovná 1 pokud jsou kladní oba sousedé nebo jeden ze sousedů je kladný a druhý zatím nemá znaménko. Pokud se hodnota $\bar{\chi}$ se rovná 0, pak oba sousedi zatím nemají znaménko nebo znaménka obou sousedů jsou vzájemně opačná. Hodnota $\bar{\chi}$ se rovná -1 pokud jsou záporní oba sousedé nebo jeden ze sousedů je záporný a druhý zatím nemá znaménko. Predikci znaménka kódovaného koeficientu udává veličina $\hat{\chi}$. Pokud je tato predikce korektní, tak na výstupu je symbol 0 v příslušném *Sign Coding* (SC) neboli kódování znaménka. Pokud není, pak na výstup symbol 1 (SC).

Tab. 4.1: Predikce znaménka a souvislosti *Sign Coding* (SC) neboli kódování znaménka.

χ^{-h}	χ^{-v}	κ^{SC}	$\hat{\chi}$
1	1	SC4	1
1	0	SC3	1
1	-1	SC2	1
0	1	SC1	1
0	0	SC0	1
0	-1	SC1	-1
-1	1	SC2	-1
-1	0	SC3	-1
-1	-1	SC4	-1

V případě koeficientu $(0, 2)$, nemá žádný soused znaménko. Tudíž se jedná o SC0 a predikce znaménka je kladná. Tato predikce není korektní, a proto je na výstup SC0 zapsán symbol 1.

Ve sloupci $j = 2$ zbývají koeficienty $(1, 2)$, $(2, 2)$ a $(3, 2)$. Vzhledem k tomu, že už kodér není v režimu *Run-Length*, tak na tyto koeficienty použijeme *Zero Coding* (ZC). Užívá se zde devět různých možností od ZC0 po ZC8. Tyto možnosti jsou založeny na známé významnosti osmi okolních sousedů (ve všech směrech), právě kódovaného koeficientu. Jsou definovány v tab. 4.2. Hodnoty κ^h, κ^v a κ^d představují hodnoty koeficientů v horizontálních, vertikálních a diagonálních sousedů, kteří již byli shledány významnými.

Tab. 4.2: Souvislosti sousedů a princip kódování s *Zero Coding* (ZC). „x“ značí jakoukoliv hodnotu.

Subpásma LL a LH			**Subpásmo HL**			**Subpásmo HH**		
κ^h	κ^v	κ^d	κ^h	κ^v	κ^d	κ^d	$\kappa^h + \kappa^v$	κ^{ZC}
0	0	0	0	0	0	0	0	ZC0
0	0	1	0	0	1	0	1	ZC1
0	0	≥ 2	0	0	≥ 2	0	≥ 2	ZC2
0	1	x	1	0	x	1	0	ZC3
0	2	x	2	0	x	1	1	ZC4
1	0	0	0	1	0	1	≥ 2	ZC5
1	0	≥ 1	0	1	≥ 1	2	0	ZC6
1	≥ 1	x	≥ 1	1	x	2	≥ 1	ZC7
2	x	x	x	2	x	≥ 3	x	ZC8

Koeficient $(1, 2)$ má pouze jednoho významného souseda a to koeficient $(0, 2)$. Mimoto koeficient $(1, 2)$ není významný v bitové hladině $n = 5$. Na výstupu (ZC3) je tedy symbol 0. Koeficienty $(2, 2)$ a $(3, 2)$ nemají ani jednoho významného souseda a nejsou významné v bitové hladině $n = 5$. Výstup (ZC0) jsou tedy dva symboly 0.

Nyní kodér kóduje třetí sloupec $j = 3$. Koeficient $(0, 2)$, který leží ve vedlejším sloupci je znám jako významný a *Run-Length* režim je vypnutý. Pro kódování je použito *Zero Coding* (ZC). Následující koeficienty se zakódují těmito symboly: 0(ZC5), 0(ZC1), 0(ZC0) a 0(ZC0).

Ve sloupci $j = 4$ nejsou známy žádné významné koeficienty. Navíc, není znám žádný významný soused (sousední koeficient). Proto je opět aktivován *Run-Length* režim. Koeficient $(1, 4)$ je v bitové hladině $n = 5$ významný, a proto je zapsán na výstup (RL) symbol 1. Na výstup je také zapsán symbol 01, jelikož kodér indikuje první významný koeficient v tomto sloupci. Poté je kódováno znaménko koeficientu.

Koeficient $(1, 4)$ je kladný. Znaménko je (SC0). Predikce znaménka je správná, proto je na výstup (SC0) zapsán symbol 0. Následující dva koeficienty jsou kódovány pomocí *Zero Coding* (ZC). Na výstup jsou zapsány tyto symboly: 0(ZC3) a 0(ZC0).

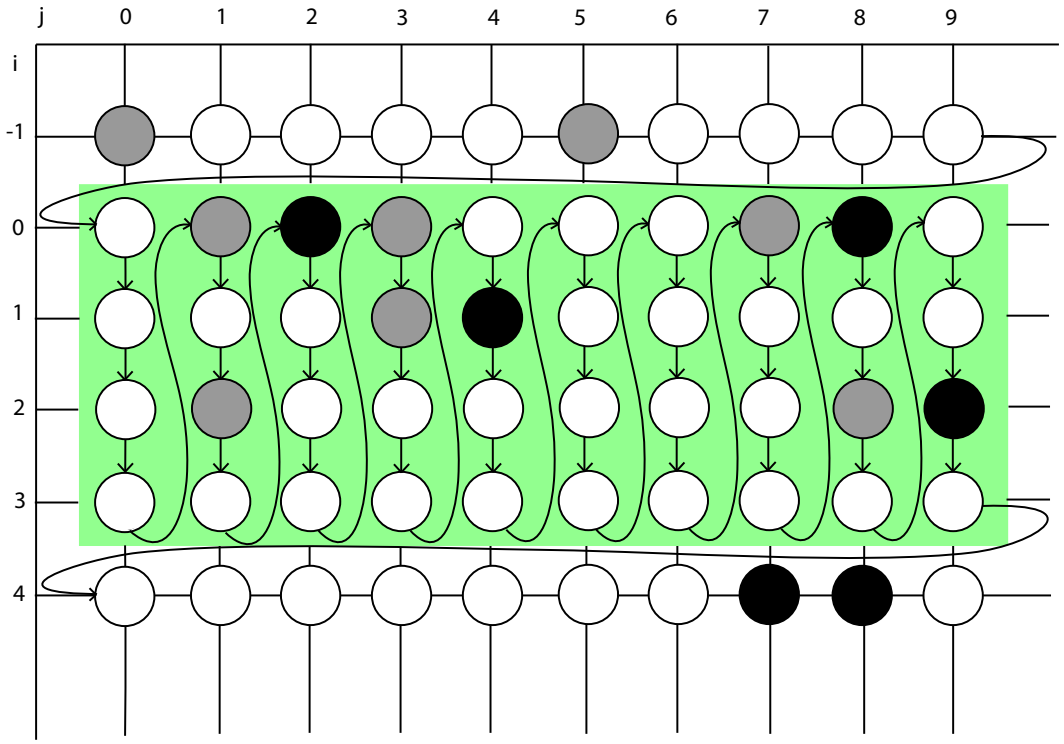
Zbývající kódování průchodu *Cleanup Pass* v bitové hladině $n = 5$ je popsáno v tab. 4.3.

Tab. 4.3: Kódování bitové hladiny na úrovni 5 pomocí průchodu *Cleanup Pass* (pro uvedený příklad).

	Kódované symboly	**Komentář**
$j = 0$	0(RL)	*Run-Length* režim.
$j = 1$	0(RL)	*Run-Length* režim.
$j = 2$	1(RL)	*Run-Length* režim do koeficientu $(0, 2)$.
	00(UNI)	Pozice prvního významného koeficientu.
	1(SC0)	Predikce znaménka v souvislosti (SC0) není korektní.
	0(ZC3) 0(ZC0) 0(ZC0)	Zbytek sloupce je zakódován pomocí *Zero Coding* (ZC).
$j = 3$	0(ZC5) 0(ZC1)	*Run-Length* režim není aktivní (pokračuje ZC).
	0(ZC0) 0(ZC0)	Koeficient $(0, 2)$ je významný.
$j = 4$	1(RL)	*Run-Length* režim do koeficientu $(1, 4)$.
	01(UNI)	Pozice prvního významného koeficientu.
	0(SC0)	Predikce znaménka v souvislosti (SC0) je korektní.
	0(ZC3) 0(ZC0)	Zbytek sloupce je zakódován pomocí *Zero Coding* (ZC).
$j = 5$	0(ZC1) 0(ZC5)	*Run-Length* režim není aktivní (pokračuje ZC).
	0(ZC1) 0(ZC0)	Koeficient $(1, 4)$ je významný.
$j = 6$	0(RL)	*Run-Length* režim.
$j = 7$	0(RL)	*Run-Length* režim.
$j = 8$	1(RL)	*Run-Length* režim do koeficientu $(0, 8)$.
	00(UNI)	Pozice prvního významného koeficientu.
	0(SC0)	Predikce znaménka v souvislosti (SC0) je korektní.
	0(ZC3) 0(ZC0) 0(ZC0)	Zbytek sloupce je zakódován pomocí *Zero Coding* (ZC).
$j = 9$	0(ZC5)	*Run-Length* režim není aktivní (pokračuje ZC).
	0(ZC1)	Koeficient $(0, 8)$ je významný.
	1(ZC0)	Významný koeficient v souvislosti (ZC0).
	1(SC0)	Predikce znaménka v souvislosti (SC0) není korektní.
	0(ZC3)	Zbytek sloupce je zakódován pomocí *Zero Coding* (ZC).

4.3.3 Kódování druhé bitové hladiny

V předchozí části jsem si podrobně rozebrali kódování první bitové hladiny, kde využíval pouze kódovací průchod *Cleanup Pass*. Nyní si přiblížíme kódování druhé bitové hladiny, kde se využívají už všechny uvedené průchody.



Obr. 4.8: Bitová hladina na úrovni 4. Šedé buňky představují koeficienty, které jsou v této hladině významné.

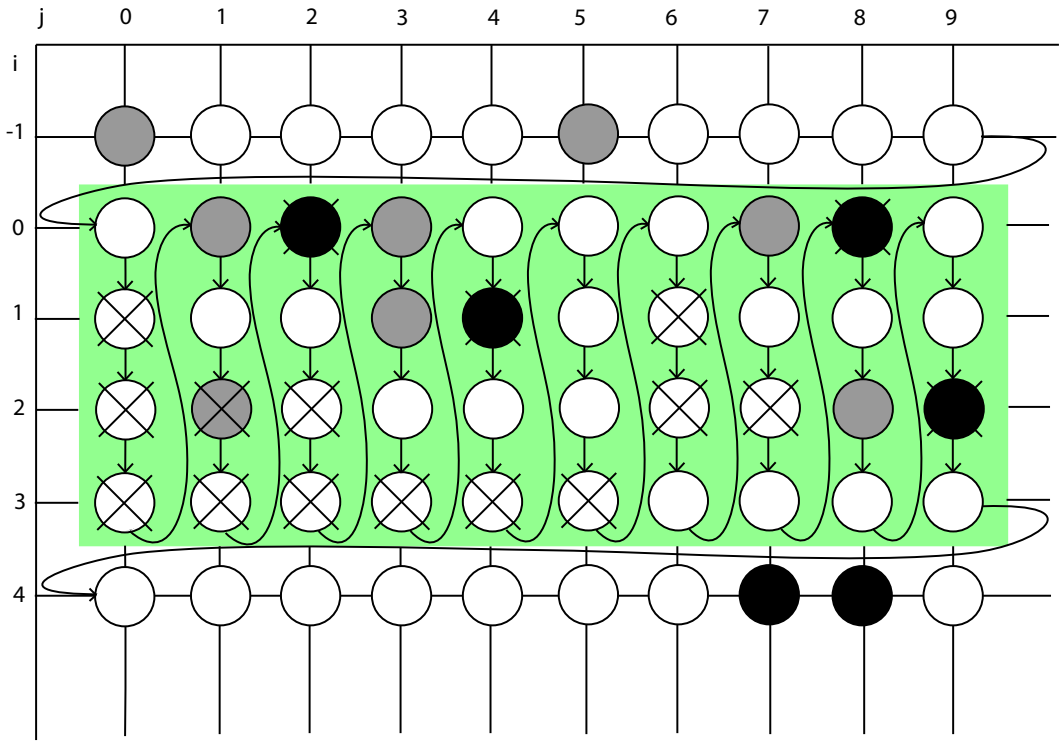
Druhá bitová hladina ($n = 4$) je zobrazena na obr. 4.8. Významné koeficienty z předchozí kódované bitové hladiny jsou zde reprezentovány černou barvou. Koeficienty, které byli shledány v této bitové hladině ($n = 4$) jsou reprezentovány šedou barvou. Jak už jsme zmínili v této druhé bitové hladině se používají všechny tři kódovací průběhy. Jako první z nich průchod *Significance Propagation* kóduje sousedy koeficientů, kteří již bylo shledány významnými v této bitové hladině ($n = 4$) nebo vyšší významnější bitové hladině (v našem případě $n = 5$). Vychází se z předpokladu, že vysoké hodnoty koeficientů se obecně vyskytují ve shlucích. Tudíž se významné koeficienty v první řadě hledají okolo vysokých hodnot koeficientů, které jsou již známy. Poté průchod *Magnitude Refinement* je použit k zpřesňování hodnot koeficientů, které byli shledány významnými a zakódovány v předchozí bitové hladině. Nakonec průchod *Cleanup Pass* zakóduje zbývající bity.

Průchod Propagation Pass

Skenování bitové hladiny $n = 4$ začíná koeficientem $(0, 0)$. Sousedem tohoto koeficientu je koeficient $(-1, 0)$ a ten se stal v bitové hladině $n = 4$ významným. Bit na pozici $(0, 0)$ je tedy kódován průchodem *Significance Propagation*. V tomto průchodu

se využívá pouze *Zero Coding* (ZC). Bit $(0, 0)$ se zakóduje a na výstup v souvislosti (ZC3) se pošle symbol 0.

Je známo, že koeficienty $(1, 0)$, $(2, 0)$ a $(3, 0)$ nemají žádného významného souseda, a proto se nebudou kódovat v průběhu *Significance Propagation*. Tyto koeficienty jsou označeny křížkem v obr. 4.9



Obr. 4.9: Průchod *Significance Propagation* v bitové hladině na úrovni 4.

Koeficient $(0, 2)$ je významný, proto jsou bity na pozici $(0, 1)$ a $(1, 1)$ zakódovány. Bit $(0, 1)$ je významný v bitové hladině $n = 4$, a proto je na výstup poslán symbol 1(ZC6). Znaménko waveletového koeficientu je zakódováno jako symbol 1 (SC3). Z toho plyne, že predikce znaménka vyšla záporná, avšak ve skutečnosti je koeficient kladný. Pro zakódování bitu $(1, 1)$ je na výstup poslán symbol 0(ZC3).

Dále koeficienty $(2, 1)$ a $(3, 1)$ nejsou kódovány. Samotný koeficient $(0, 2)$ není kódován, protože byl významný v předchozí bitové hladině. Bit $(2, 1)$ se zakóduje tím, že se pošle na výstup symbol 0(ZC3). Následující koeficienty $(2, 2)$ a $(3, 2)$ nejsou kódovány.

Zbývající kódování bitové hladiny $n = 4$ průběhem *Significance Propagation* je popsáno v tab. 4.4.

Tab. 4.4: Kódování bitové hladiny na úrovni 4 pomocí průchodu *Significance Propagation* (pro uvedený příklad).

	Kódované symboly	**Komentář**
$j = 0$	0(ZC3)	Je zakódován koeficient $(0, 0)$.
$j = 1$	1(ZC6)	Je zakódován koeficient $(0, 1)$.
	1(SC3)	Predikce znaménka v (SC3) není korektní.
	0(ZC3)	Je zakódován koeficient $(1, 1)$.
$j = 2$	0(ZC3)	Je zakódován koeficient $(1, 2)$.
$j = 3$	1(ZC6) 0(SC3)	Je zakódován koeficient $(0, 3)$.
	1(ZC7) 0(SC2)	Koeficient $(1, 3)$.
	0(ZC3)	Koeficient $(2, 3)$.
$j = 4$	0(ZC7) 0(ZC3)	Koeficienty $(0, 4)$ a $(2, 4)$.
$j = 5$	0(ZC3) 0(ZC5) 0(ZC1)	Koeficienty $(0, 5)$ a $(2, 5)$.
$j = 6$	0(ZC1) 0(ZC1)	Koeficienty $(0, 6)$ a $(3, 6)$.
$j = 7$	1(ZC5) 1(SC3)	Koeficient $(0, 7)$.
	0(ZC3) 0(ZC3)	Koeficienty $(1, 7)$ a $(3, 7)$.
$j = 8$	0(ZC3)	Koeficient $(1, 8)$.
	1(ZC5) 1(SC3)	Koeficient $(2, 8)$.
	0(ZC4)	Koeficient $(3, 3)$.
$j = 9$	0(ZC5) 0(ZC3) 0(ZC3)	Koeficienty $(0, 9)$, $(2, 9)$ a $(3, 9)$.

Průchod Magnitude Refinement

Tento průchod používá pouze kódování *Magnitude Refinement* (MR). Tento průchod kóduje bity, které jsou již významné v předchozí bitové hladině. Definují se tři typy kódování, které jsou podrobně vysvětleny v tab. 4.5. Jsou označeny jako MR0, MR1 a MR2. Pokud je veličina $\tilde{\sigma}$ rovna hodnotě 0, tak se jedné o to, že byl poprvé aplikován průchod *Magnitude Refinement* na kódovaný koeficient. Jinak se veličina $\tilde{\sigma}$ rovna hodnotě 1. V případě veličin κ^h a κ^v, tak nesou stejný význam jako v případě *Zero Coding* (ZC).

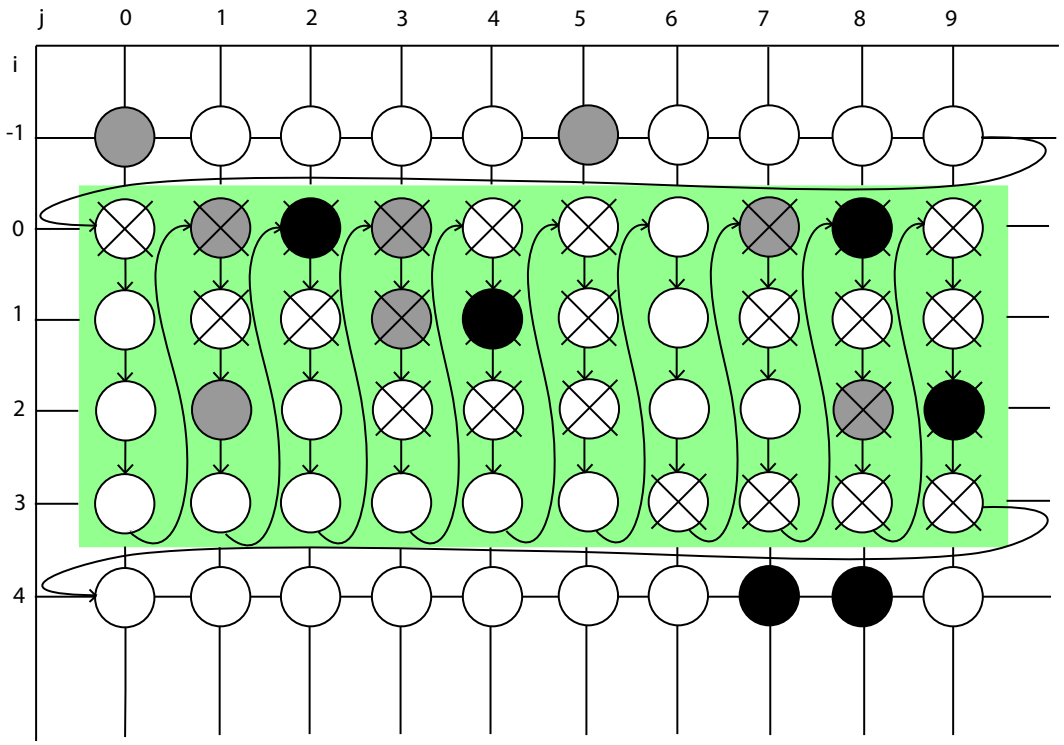
Tab. 4.5: Typy znaků kódování *Magnitude Refinement* (MR).

$\tilde{\sigma}$	$\kappa^h + \kappa^v$	κ^{MR}
0	0	MR0
0	$\neq 0$	MR1
1	x	MR2

Hodnoty koeficientů, které byly významné v předchozí bitové hladině jsou zpřes-

ňovány v tomto průběhu. Symbol 0(MR1) je poslán na výstup v případě zpřesnění koeficientu $(0, 2)$. Stejným způsobem jsou symboly 1(MR1). 0(MR1) a 1(MR1) poslány na výstup k upřesnění koeficientů $(1, 4)$, $(0, 8)$ a $(2, 9)$.

Průchod Cleanup Pass



Obr. 4.10: Průchod *Cleanup Pass* v bitové hladině na úrovni 4.

Průchod *Cleanup Pass* je použit je kódování bitů, které nebyli zakódovány v předchozích dvou průbězích. Na obr. 4.10 tyto bity nejsou označeny křížkem. Kódovací průběh je popsán v tab. 4.6.

4.3.4 Pokračování a ukončení kódování

Dosud jsme si popsali kompletně kódování první dvou bitových hladin (nejvyšších). Celý tento proces se opakuje do té doby, dokud hodnota bitové hladiny není $n = 0$. Nakonec je výstup kodéru podroben vhodnému aritmetickému kódování, které využívá pravděpodobností jednotlivých symbolů. Tyto pravděpodobnosti jsou aktualizovány po každém zakódovaném symbolu. Tento kódovací proces kódovacích bloků je první částí samotného EBCOT kodéru, zvaný jako Tier1 (Stupeň 1). Ve Tier2 (Stupni 2) jsou části výstupního bitového toku (po jednotlivých bitových hladinách)

Tab. 4.6: Kódování bitové hladiny na úrovni 4 pomocí průchodu *Cleanup Pass* (pro uvedený příklad).

	Kódované symboly	**Komentář**
$j = 0$	0(ZC1) 0(ZC0) 0(ZC0)	
$j = 1$	1(ZC0) 0(SC0) 0(ZC3)	Koeficient $(2, 1)$.
$j = 2$	0(ZC6) 0(ZC1)	
$j = 3$	0(ZC0)	
$j = 4$	0(ZC0)	
$j = 5$	0(ZC0)	
$j = 6$	0(ZC6) 0(ZC1) 0(ZC0)	
$j = 7$	0(ZC5)	

kodéru organizovány tak, aby byl výsledný bitový tok optimalizován jako kompromis vůči zkreslení. Celý tento proces se nazývá *Post-Compression Rate-Distortion OPTimization*, zkráceně pak jako PCRD-opt.

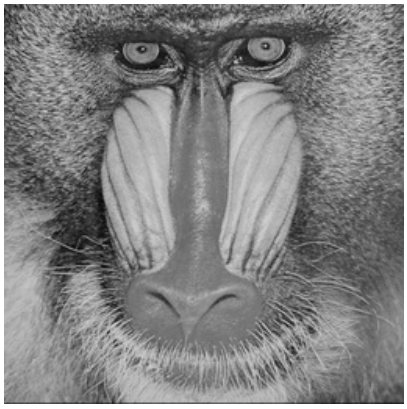
5 VYHODNOCENÍ

V této kapitole si vyhodnotíme jednotlivé kodeky (kódovací algoritmy) přímo na příkladech. Následně je vyhodnotíme pomocí kvantitativních a kvalitativních metod (PSNR, SSIM, bpp), které jsme v této práci shrnuli.

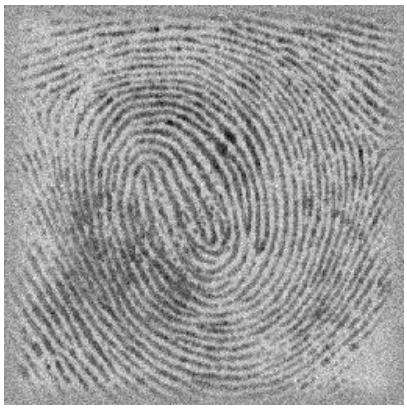
Součástí práce jsou i implementované kodeky EZW a SPIHT (v prostředí `Matlab`). Z hlediska složitosti jsou přizpůsobeny pouze pro obrazy ve stupních šedi a o rozměrech N × N. V naší ukázce je otestujeme na třech různých obrázcích a vyhodnotíme jejich výslednou kvalitu.

Jako první testovací obrázek byl zvolen *lena256.bmp* o rozměrech 256 × 256, který je součástí prostředí `Matlab`. Samozřejmě budeme uvažovat obraz ve stupních šedi. Dalšími obrázky jsou pak*baboon256.bmp* a $finger256.bmp$, které jsou rovněž součástí prostředí `Matlab`. Všechny testovací obrázky jsou zobrazeny na obr. 5.1.

(a) lena256



(b) baboon256



(c) finger256

Obr. 5.1: Testovací obrázky

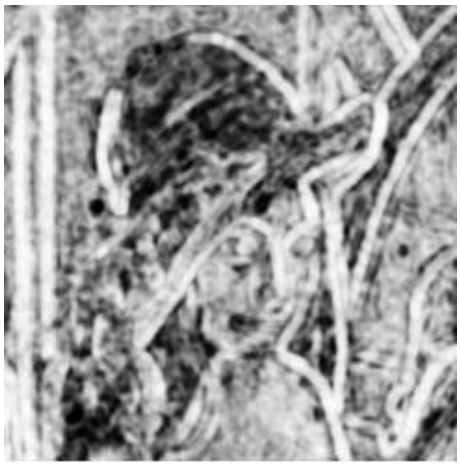
Nyní budou následovat komprese jednotlivých obrázku pomocí EZW a SPIHT. Na obr. 5.2, 5.3, 5.4 a 5.5, můžeme vidět srovnání těchto dvou kodeků. U obrázku

lena256 jsou převedeny výsledky při dvou různých hodnotách bpp. Dále jsou zde zobrazeny jak samotné výstupní obrázky obou algoritmů, tak jejich tzv. *ssim index mapa*. Toto zobrazení, závisí na hodnotě SSIM indexu a udává míru změny. Kdyby byla zobrazena tato *mapa* pro dva totožné obrazy, tak výsledkem by byla zcela bílá plocha. Černá barva značí změnu. Z tohoto vyplývá, že obecně čím světlejší *ssim index mapa* tím lepší výsledek pro pozorovatele. Nakonec jsou uvedeny samotné hodnoty bpp, PSNR a SSIM index.

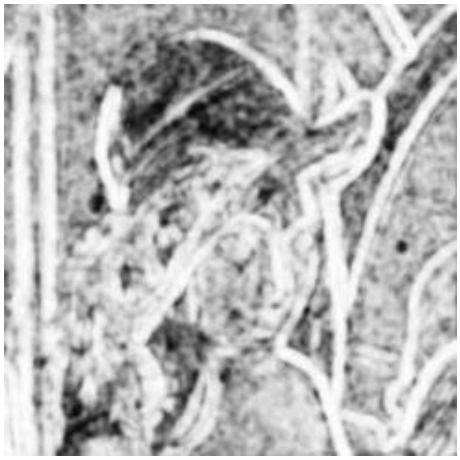
Výchozí neměnnou hodnotou je bpp. Díky tomu lze názorně porovnat, jak se mění hodnoty PSNR a SSIM v jednotlivých kodecích. Na základě těchto výsledků lze vyvodit závěr, že algoritmus SPIHT je na tom viditelně lépe a dosahuje lepších výsledků při nižších hodnotách PSNR a vyšších hodnotách SSIM indexu.

(a) PSNR = 30,26 dB, SSIM = 0,8644

(b) PSNR = 33,04 dB, SSIM = 0,9101



(c) SSIM index mapa (EZW)



(d) SSIM index mapa (SPIHT)

Obr. 5.2: Výsledky testovaného obrázku *lena256* při hodnotě bpp = 0,64. Kodek EZW (vlevo) a SPIHT (vpravo)

(a) PSNR = 26,60 dB, SSIM = 0,7721

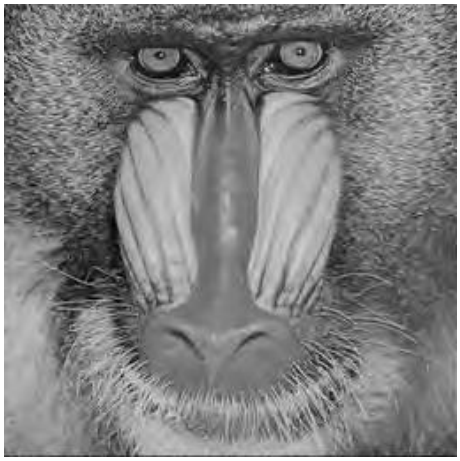
(b) PSNR = 28,48 dB, SSIM = 0,8310

(c) SSIM index mapa (EZW)

(d) SSIM index mapa (SPIHT)

Obr. 5.3: Výsledky testovaného obrázku *lena256* při hodnotě bpp = 0,29. Kodek EZW (vlevo) a SPIHT (vpravo)

(a) PSNR = 26,73 dB, SSIM = 0,7627



(b) PSNR = 28,40 dB, SSIM = 0,8401

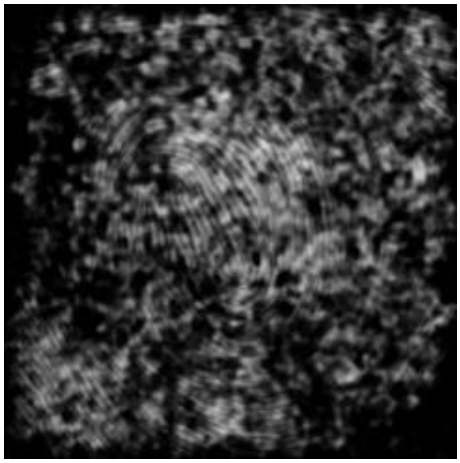
(c) SSIM index mapa (EZW)

(d) SSIM index mapa (SPIHT)

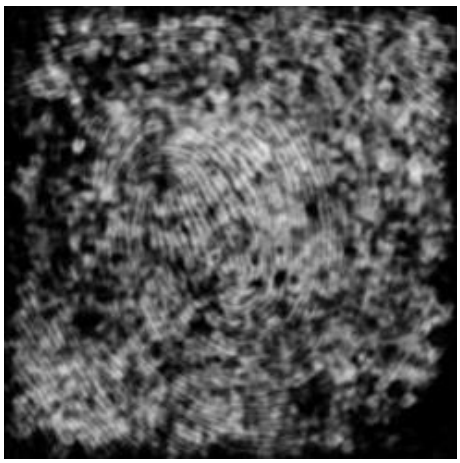
Obr. 5.4: Výsledky testovaného obrázku *baboon256* při hodnotě bpp = 0,99. Kodek EZW (vlevo) a SPIHT (vpravo)

(a) PSNR = 21,37 dB, SSIM = 0,5849

(b) PSNR = 22,27 dB, SSIM = 0,6906



(c) SSIM index mapa (EZW)



(d) SSIM index mapa (SPIHT)

Obr. 5.5: Výsledky testovaného obrázku *finger256* při hodnotě bpp = 0,44. Kodek EZW (vlevo) a SPIHT (vpravo)

6 ZÁVĚR

V této práci jsem představil základní poznatky reprezentace barev v počítači, přičemž jsem charakterizoval barevný model RGB, LAB a YC_BC_R. Dále jsem vysvětlil, co je myšleno pod pojmem paletové vyjádření barev True Color vyjádření. V neposlední řadě jsem uvedl základní vlastnosti bitmapových obrazových formátů a zhodnotil výhody a nevýhody formátů JPEG a JPEG2000.

Rovněž jsem shrnul poznatky základů datové komprese a komprese obrazů, kde jsem charakterizoval typické kódování využívané jak v bezeztrátové kompresi, tak v kompresi ztrátové. Současně jsem uvedl základní typy transformačního kódování, následně je stručně představil a doplnil o poznatky a způsoby hodnocení kvality výsledného obrazu, jež se využívají při kvalitativním i kvantitativním hodnocení.

Ve své práci jsem rovněž představil základy vlnkové (waveletové) transformace, dále pak především její diskrétní formu DWT a dvourozměrnou formu 2D DWT. Zabýval jsem se rovněž pojmem dekompozice obrazu, který jsem názorně popsal a předvedl a uvedl některé vlnky (wavelety) používané při obrazové kompresi.

Podrobně jsem se zabýval obrazovými kodeky založenými na vlnkové transformaci obrazu (EZW, SPIHT, EBCOT) a uvedl jejich podstatu a princip. Následně jsem implementoval kodeky EZW a SPIHT v prostředí `MATLAB`. Kodek zvaný EBCOT nebyl implementován. Důvodem byl nedostatek vhodných materiálů a vysoká složitost jeho implementace samotné.

Závěrem mé práce jsem otestoval kodeky EZW a SPIHT na třech různých obrázcích a vyhodnotil jejich úspěšnost pomocí způsobů hodnocení kvality výsledného obrazu (bpp, PSNR, SSIM). Z výsledků je více než patrné že algoritmus SPIHT má značně navrch oproti staršímu EZW, a tudíž je efektivnějším kódovacím algoritmem.

LITERATURA

[1] ČÍKA, P. *Multimediální služby*. Brno: Vysoké učení technické v Brně, 2012. s. 127. ISBN: 978-80-214-4443-0.

[2] DELAUNAY, X. *EBCOT coding passes explained on a detailed example* [online]. [cit. 6. 12. 2012]. Dostupné z URL: <`http://d.xav.free.fr/ebcot/EBCOT_example.pdf`>.

[3] DELAUNAY, X., CHABERT, M., MORIN, G., CHARVILLAT, V. *Bit-plane analysis and contexts combining of JPEG2000 contexts for on-board satellite image compression* [online]. 2007, [cit. 6. 12. 2012]. Dostupné z URL: <`http://d.xav.free.fr/bib/icassp07_paper_delaunay.pdf`>.

[4] HANUS, S. *Základy televizní techniky III*. Skriptum FEKT VUT v Brně. Brno, MJ Servis, 2010. 103 s. ISBN 978-80-214-4206-1.

[5] KINTL, V. *Využití vlnkové transformace při kompresi videosignálu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 79 s. Vedoucí diplomové práce Ing. Jan Malý.

[6] KUČERA, M. *Segmentovaná vlnková transformace obrazu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2010. 51 s. Vedoucí diplomové práce Ing. Zdeněk Průša

[7] KVÁŠ, M. *Komprese JPEG 2000 a akcelerace pomocí DSP* [online]. 9. 4. 2008, roč. 2008, č. 13, s. 12 [cit. 6. 12. 2012]. ISSN 1213-1539. Dostupné z URL: <`http://elektrorevue.cz/cz/clanky/zpracovani-signalu/35/komprese-jpeg-2000-a-akcelerace-pomoci-dsp/`>.

[8] MALÝ, J. *Srovnání metod pro ztrátovou kompresi obrazu* [online]. 25. 10. 2006, roč. 2006, č. 42, s. 11 [cit. 6. 12. 2012]. ISSN 1213-1539. Dostupné z URL: <`http://www.elektrorevue.cz/clanky/06042/index.html`>.

[9] MOKREJ, J. *Vlnková komprese obrazu*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2008. 46 s. Vedoucí bakalářské práce Ing. Kamil Říha, Ph.D.

[10] NĚMEC, K., ŠILHAVÝ, P. *Datová komunikace*. Brno: Vysoké učení technické v Brně, 2011. s. 205.

[11] PRŮŠA, Z., MALÝ, J. *Metoda SPIHT pro kompresi barevných obrazů a její implementace* [online]. 6. 4. 2009, roč. 2009,

č. 17, s. 11 [cit. 6. 12. 2012]. ISSN 1213-1539. Dostupné z URL: <http://www.elektrorevue.cz/cz/clanky/zpracovani-signalu/30/metoda-spiht-pro-kompresi-barevnych-obrazu-a-jeji-implementace/>.

[12] RAJMIC, P. *Využití waveletové transformace a matematické statistiky pro separaci signálu a šumu*. Dizertační práce, FEKT VUT v Brně, 2004.

[13] RAJMIC, P. *Základy počítačové sazby a grafiky*. Brno: Vysoké učení technické v Brně, 2012. s. 160. ISBN: 978-80-214-4451-5.

[14] RAO, Raghuveer M., BOPARDIKAR, Ajit S. *Wavelet Transforms Introduction to Theory and Applications*. Addison Wesley Longman, 1998. 310 s. ISBN 0-201-63463-5.

[15] SAID, A., PEARLMAN W.A. *A new fast and efficient image codec based on set partitioning in hierarchical trees*. IEEE Transactions on Circuits and Systems for Video Technology 6 (3): 243–250, 1996

[16] SAYOOD, K. *Introduction to Data Compression*. 2nd ed. Morgan Kaufmann Publishers, 2000. 636 s. ISBN 1-55860-558-4.

[17] TAUBMAN, David S., MARCELLIN, Michael W. *JPEG2000 Image Compression Fundamentals, Standarts and Practice*. Kluwer Academic Publishers, 2002. 773 s. ISBN 0-7923-7519-X.

[18] WANG, Z., BOVIK, A. C. *Mean squared error: love it or leave it? - A new look at signal fidelity measures* [online]. IEEE Signal Processing Magazine, vol. 26, no. 1, pp. 98-117, Jan. 2009. [cit. 6. 12. 2012]. Dostupné z URL: <https://ece.uwaterloo.ca/~z70wang/publications/SPM09.pdf>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

2D DWT dvourozměrná vlnková transformace – 2D Discrete Wavelet Transform

BMP vzorkovací kmitočet

bpp počet bitů na pixel – Bits Per Pixel

CDF vzorkovací kmitočet

CWT integrální vlnková transformace – Continuous Wavelet Transform

DCT diskrétní kosinova transformace – Discrete Cosine Transform

DTWT diskrétní vlnková transformace s diskrétní časem

DWT diskrétní vlnková transformace – Discrete Wavelet Transform

EBCOT obrazový kodek – Embedded Bitplane Coding with Optimal Truncation

EZW kódovací algoritmus – Embedded Zerotree Wavelet

FFT rychlá Fourierova transformace – Fast Fourier transform

FIR filtr s konečnou impulzní odezvou – Finite impulse response

GIF obrazový formát – Graphics Interchange Format

HSB barevný model – Hue, Saturation, Brightness

HTML značkovací jazyk – HyperText Markup Language

JPEG obrazový formát – Joint Photographic Experts Group

JPEG2000 obrazový formát – Joint Photographic Experts Group 2000

LAB barevný prostor – CIE 1976 (L*, a*, b*)

MSE střední kvadratická chyba – Mean squared error

PNG obrazový formát – Portable Network Graphics

PCRD-opt Post-Compression Rate-Distortion OPTimization

PSNR špičkový poměr signálu k šumu – peak signal-to-noise ratio

RGB barevný model

RGBA barevný model RGB s alfa kanálem pro průhlednost

RLE Run-length encoding

SPITH kódovací algoritmus – Set Partitioning in Hierarchical Trees

SSIM structural similarity index

YC_BC_R barevný model používaný u videa a nebo u digitální fotografie

YUV barevný model používaný v televizním vysílání

A OBSAH PŘILOŽENÉHO DVD

Přiložené DVD obsahuje elektronickou verzi bakalářské práce. Hlavní dokument je nazván „`Obrazové kodeky založené na vlnkové transformaci`“. Dále ve složce „`Implementace`“ jsou v jednotlivých podadresářích (`EZW` a `SPIHT`) implementovány kodeky. Ukázky jednotlivých kodeků se spouští pomocí souborů `ukazka_EZW.m` nebo `ukazka_SPIHT.m`. Po otevření si uživatel zvolí parametry komprese a spustí zmíněný `M-file`.

Obsah:

- `Obrazové kodeky založené na vlnkové transformaci.pdf`
- `Implementace`
  - `EZW`
  - `SPIHT`