VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMPRESE OBRAZU POMOCÍ VLNKOVÉ TRANSFORMACE

IMAGE COMPRESSION USING THE WAVELET TRANSFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. VÁCLAV BRADÁČ**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. DAVID BAŘINA, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání diplomové práce

Řešitel: **Bradáč Václav, Bc.**
Obor: Počítačová grafika a multimédia
Téma: **Komprese obrazu pomocí vlnkové transformace**
Image Compression Using the Wavelet Transform
Kategorie: Zpracování obrazu

Pokyny:

1. Seznamte se s algoritmy používanými pro ztrátovou i bezeztrátovou kompresi statického obrazu pomocí vlnkové transformace.
2. Navrhněte postup komprese obrazu pomocí této transformace.
3. Implementujte navržený postup jako knihovnu v programovacím jazyce C nebo C++.
4. Srovnejte účinnost komprese při užití různých parametrů komprese. Porovnejte dosaženou účinnost komprese také se v současnosti používanými formáty.
5. Diskutujte výsledky Vaší práce a potenciál použití vlnkové transformace ke kompresi obrazu.

Literatura:

- S. Mallat, A Wavelet Tour of Signal Processing : The Sparse Way. 3rd edition. Academic Press, c2009. xx, 805 s. ISBN 9780123743701.
- 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, ročník 6, č. 3, 1996.

Při obhajobě semestrální části projektu je požadováno:

- První dva body zadání.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci dřívějších projektů (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bařina David, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 24. května 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Tato práce se zabývá kompresí obrazu s využitím vlnkové transformace. Na jejím začátku lze nalézt teoretické informace o nejznámějších postupech, které se využívají pro kompresi obrazu, důkladný popis vlnkové transormace a algoritmu EBCOT. Značná část práce je věnována vlastní implementací knihovny. Další kapitola diplomové práce se zabývá srovnáním a vyhodnocením dosažených výsledků zpracované knihovny s formátem JPEG2000.

Abstract

This work deals with image compression using wavelet transformation. At the beginning, you can find theoretical information about the best known techniques used for image compression, a thorough description of wavelet transormation and the EBCOT algorithm. A significant part of the work is devoted to the library's own implementation. Another chapter of the diploma thesis deals with the comparison and evaluation of the achieved results of the processed library with the JPEG2000 format.

Klíčová slova

Komprese obrazu, diskrétní vlnková transformace, konturletova transformace, EZW, SPIHT, EBCOT, kódování, PSNR, SSIM, aritmetický kodér, lifting

Keywords

Image compresion, discrete wavelet transform, contourlet transform, EZW, SPIHT, EBCOT, coding, PSNR, SSIM, arithmetic coding, lifting

Citace

BRADÁČ, Václav. *Komprese obrazu pomocí vlnkové transformace.* Brno, 2017. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bařina David.

Komprese obrazu pomocí vlnkové transformace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Václav Bradáč
23. května 2017

Poděkování

Tímto bych rád poděkoval svému vedoucímu Ing. David Bařina, Ph.D. za odborné vedení a podněty, které mi při řešení projektu poskytl.

Obsah

Kapitola 1

Úvod

Navzdory rychlému pokroku ve výkonu systému, rychlosti procesoru a velikosti úložišť, jsou neustále větší nároky na snižování velikostí multimediálních dat pro uložení co nejvíce datových informací na disk nebo ke snížení šířky přenosového pásma. Komprese dat, zvláště komprese obrazu, hraje velmi důležitou roli v oboru multimediálních počítačových služeb a dalších telekomunikačních aplikací. Oblast obrazové komprese má široké spektrum metod od klasických bezztrátových až po novější populární transformační přístupy založených na segmentačních kódovacích metodách. Až donedávna byla diskrétní kosinová transformace nejpopulárnější technikou komprese, kvůli optimálnímu výkonu a snadné realizaci za přiměřenou cenu. Několik komerčně úspěšných algoritmů komprese, jako je standard JPEG pro statické obrázky a standart MPEG pro snímky videa, jsou založeny právě na diskrétní kosinové transformaci.

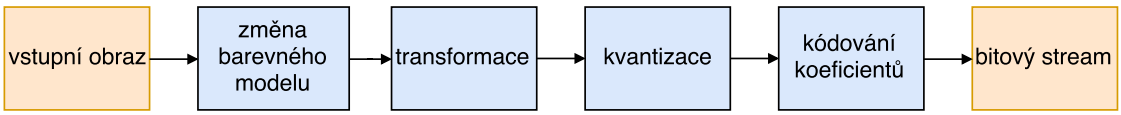
Techniky založené na vlnkové transformaci jsou nyní nejvíce využívané při vývoji v oblasti komprese obrazových dat, protože nabízí schopnost multirezoluce, která vede k vynikajícímu konsolidaci energie s vysoce kvalitními rekonstruovanými obrazy při vysokých kompresních poměrech. Vlnková transformace se objevila jako špičková technologie v oblasti komprese obrazu. Tato transformace převzala otěže od kosinové transformace kvůli tomu, že vyřešila problém s tvorbou artefaktů, které se při její kompresi objevovali. Navíc vlnková transformace snižuje korelaci mezi sousedními pixely a poskytuje vícenásobné rozložení obrazu. Během posledních několika let byla vyvinuta a implementována řada výkonných a sofistikovaných systémů, založených na vlnkové transformaci právě pro kompresi obrazu. Nový standard ISO/ITU-T pro kódování statických obrazových dat JPEG2000 je kompresní algoritmus založený na vlnkách. Tento algoritmus je navržen, aby řešil požadavky různých druhů aplikací včetně internetu, digitálních fotografií, obrazových databází, zobrazování dokumentů, mobilních aplikací, lékařských snímků a dalších.

Diplomová práce zahrnuje část teoretickou, praktickou a srovnání dosažených výsledků. V teoretické části o kompresi obrazu jsou mimo jiné popsány barevné modely, metriky pro zjištění kvality komprese, transformace a metody kódování transformovaných koeficientů. V praktické části je popsána samotná implementace knihovny. V sekci práce, která obsahuje srovnání jednotlivých transformací a metod kódování koeficientů vzniklých transformací je vysvětleno, jakým způsobem jsou výsledky ovlivněny nastavením provedení algoritmu. Na samotném závěru diplomové práce dochází k vyhodnocení dosažených výsledků a její celkový přínos.

Kapitola 2

Komprese obrazu

Komprese obrazu slouží k odstranění nebo redukci redundantních informací z digitálního obrazu. Tohoto jevu se využívá pro lepší využití prostoru v paměti. Existují dva základní způsoby reprezentace obrazových informaci v počítačové grafice. Prvním z nich se nazývá vektorový, který je složen ze základních přesně definovaných tvarů jako jsou body, přímky a křivky. Výsledný obraz se následně skládá z množiny těles. Vektorová grafika obsahuje malou míru redundance. Její snížení velikosti souboru spočívá v kompresi uložení samotných objektů. Druhý způsob reprezentace jakým počítače ukládají a zpracovávají obrazové informace, se nazývá rastrový. Ten popisuje obrázek pomocí jednotlivých barevných bodů (pixelů). Body jsou uspořádány do pravidelné mřížky. Každý bod má určenou svoji polohu v obraze a barvu ve zvoleném barevném modelu. Oproti vektorové reprezentaci je rastrová mnohem paměťově náročnější, a proto se u ní více využívá komprese. Tato práce se zabývá kompresí rastrových obrazových dat. Obecný princip kompresního řetězce pro rastrové obrázky je znázorněn na obrázku 2.1.



Obrázek 2.1: Blokový popis principu obecného kompresního řetězce pro rastrové obrázky.

2.1 Barevné modely

Správná volba barevného modelu již může vést ke kompresi obrazu. Lze využít vlastností lidského vnímání a pomocí něho lze určit důležité a nedůležité složky barev. Nedůležité

složky mohou být podvzorkovány. Lidský zrak je citlivější více na změnu jasové složky než na změnu barvy. Většina zobrazovacích zařízení ale využívá RGB, které funguje pomocí aditivního skládání rovnocenných tří barev. Existuje více barevných modelů, některé jsou vhodné pro kompresi obrazu a jiné pro jeho zpracování.

RGB

Barevný model RGB je založený na aditivním míchání barev. Každá barva je složena ze tří základních barev světla a to červené, zelené a modré. V zobrazovacích zařízeních se nejvíce setkáme s podporou RGB24. Jedná se o RGB model s pevně vyznačenou celkovou velikostí všech tří složek, kde každé barvě přísluší 8 bitů v paměti. U RGB24 má každá složka 256 hodnot. Smícháním všech tří barevných složek získáme 256^3 tedy 16,7 miliónů různých barev. Existují i jiné verze RGB například s více bity na jednu složku než osm, ale ty se moc nerozšířily, protože nebyla moc velká podpora od výrobců monitorů. Dále bylo vyvinuto rozšíření RGB a to RGBA, kde pod čtvrtou složkou nalezneme alfa kanál, který označuje průhlednost dané barvy pixelu.

YCBCR

YCBCR je barevný model, který zohledňuje vlastnosti lidského zraku. Skládá se ze tří složek a to jasové, červené chrominanční a modré chrominanční. Analogová verze se označuje jako YPBPR. Jedná se o formu kódování RGB informace. Vztahem (2.1) jsou definovány převody, dle standardu JFIF, mezi barevnými modely RGB24 a YCBCR.

$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0 \\ 128 \\ 128 \end{bmatrix} + \begin{bmatrix} 0{,}299 & 0{,}587 & 0{,}114 \\ -0{,}168736 & -0{,}331264 & 0{,}5 \\ 0{,}5 & -0{,}418688 & -0{,}081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.1}$$

2.2 Kvalita komprese

Vzhledem k potřebě určení kvality komprese obrazu lze využít různé metriky a metody. Máme dvě základní skupiny metrik. První z nich se nazývá subjektivní metrika, která využívá hodnocení reálných pozorovatelů. Tento postup je relativně drahý. Využívanějším tipem je druhá skupina - objektivní, která se získá pomocí výpočtu předem definovaných parametrů a jasně definovaných vztahů. V práci budou využity objektivní metody, proto jim je věnována následující podkapitola.

Objektivní metriky

Objektivní metriky popisují kvalitu obrazu pomocí různě složitých algoritmů. Velkou výhodou těchto metrik je jednoduchá implementace a rychlejší vyhodnocení. Konkrétně se dají tyto metriky rozdělit na tří základní skupiny, a to metody s plnou referencí, částečnou referencí a bez reference. První jmenovaná metoda potřebuje pro vyhodnocení celý původní obraz. Druhá metoda využívá pro svůj výpočet alespoň vybranou část původního signálu. Poslední vypsaná metoda musí mít k dispozici pouze výstupní zarušené frekvence. V následujících kapitolách jsou popsány pouze zástupci, kteří potřebují plnou referenci a to jmenovitě PSNR a SSIM.

PSNR (Peak Signal to Noise Ratio)

PSNR patří mezi objektivní, plně referenční metriky. Patří mezi nejjednodušší a nejrozšířenější metody pro určení kvality komprese. Mezi výhody patří malá výpočetní náročnost, a díky tomu je velmi využívaná. Odvozuje se od střední kvadratické chyby pixelů. Mezi její nevýhody patří, že při výpočtu nejsou zohledněny vlastnosti lidského zraku. Vzhledem k tomu, že výstupní obraz může mít vysoký a dynamický rozsah zarušení, je PSNR udáváno v decibelech (dB). Většinou jsou výsledné hodnoty u komprimovaný obrazových dat v rozmezí 20 až 50 dB, přičemž hodnoty kolem 50 dB znamenají vysokou obrazovou kvalitu.

Pro výpočet PSNR se využívá střední výpočetní chyby MSE. Podle rovnice (2.2), kde O je původní obraz a C znázorňuje komprimovaný obraz lze uskutečnit výpočet MSE.

$$\mathrm{MSE}(O,C) = \frac{1}{mn}\sum_{x=0}^{m-1}\sum_{y=0}^{n-1}[O(x,y) - C(x,y)]^2 \tag{2.2}$$

Pomocí vztahu (2.3)se vypočítá hodnota PSNR v decibelech,

$$\mathrm{PSNR}(O,C) = 10 \cdot \log_{10} 10\frac{2^{bitdepth}-1}{\mathrm{MSE}(O,C)} \tag{2.3}$$

kde m je maximální hodnota pixelu 255 pro RGB24. Z uvedených vztahů vyplývá, že metriku lze chápat jako poměr špičkového výkonu a šumu. Tato metoda nelze využít pro stejné obrázky, protože by zde docházelo k dělení nulou.

SSIM (Structural Similarity Index)

SSIM se řadí mezi objektivní a plně referenční metriky pro určení kvalitu komprese. Na rozdíl od předchozího PSNR bere částečně v úvahu model lidského vidění. Základním stavebním kamenem této metriky byla transformace výpočtu MSE tak, aby chyby v obraze byly vyhodnoceny podle míry rušivé emoce pro pozorovatele. SSIM vychází z toho, že lidský zrak je značně senzitivní na strukturu pozorované oblasti. Vztah pro funkci porovnání jasu vychází z toho, že lidský zrak je citlivější na relativní změnu jasu více, než na absolutní změnu. Při stejné změně jasu se lidské oko zaměří více na jas v pozadí scény. Obdobně jako u funkce porovnání kontrastu. U barevného obrázku se počítá pouze s jasovou složkou. Nakonec se realizuje výpočet pro porovnání struktur. Rovnicí (2.4) se vypočítá SSIM,

$$SSIM(x,y) = l(x,y)^{\alpha} \cdot c(x,y)^{\beta} \cdot s(x,y)^{\gamma} \tag{2.4}$$

kde pod l je funkce pro srovnání jasu (2.5), pod c je funkce pro srovnání kontrastu (2.6) a pod s se skrývá funkce pro porovnání struktur scény(2.7).

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.5}$$

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.6}$$

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x \cdot \sigma_y + C_3} \tag{2.7}$$

Pro hodnoty $\alpha = \beta = \gamma = 1$ a $C3 = \frac{C_2}{2}$ dostaneme výraz (2.8) pro výpočet SSIM.

$$SSIM(x,y) = \frac{2\mu_x\mu_y + C_1}{(\mu_x^2 + \mu_y^2 + C_1)\cdot(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.8}$$

kde α, β, γ jsou kladné váhovací koeficienty a μ_x, μ_y jsou střední hodnoty intezity pixelu, σ_x^2, σ_y^2 jsou rozptyly x a y, σ_{xy} jsou kovariance x a y, $C_i = (K_i L)^2$, kde L je maximální hodnota pixelu v obraze a K jsou malé konstanty obvykle udávané K_1=0,01 a K_2=0,03. Dle [7] a [17] není výsledná hodnota SSIM na těchto konstantách ve velké míře závislá.

2.3 JPEG

Existuje mnoho standardů a formátů pro ukládání obrazových dat. Tím nejrozšířenějším je formát JFIF, který je založen na JPEG kompresi. Hlavními výhodami komprese pomocí JPEG standardu je, že umožňuje dosáhnout velkých kompresních poměrů za cenu minimální ztráty vizuální kvality a má nízkou výpočetní náročnost. Parametry, které si při ukládání obrázků pomocí JPEG komprese lze zvolit, jsou mód komprese, kvalita a metodu uložení barev. Nejpoužívanějším módem komprese je sekvenční neboli základní. Dalším je progresivní mód. Při tomto způsobu se obrázek ukládá tak, že je ho možno obnovovat celý už z počátečního úseku souboru, ale se sníženou kvalitou. Mezi další možnosti se řadí bezeztrátová komprese, která není moc využívána.

Diskrétní kosinová transformace

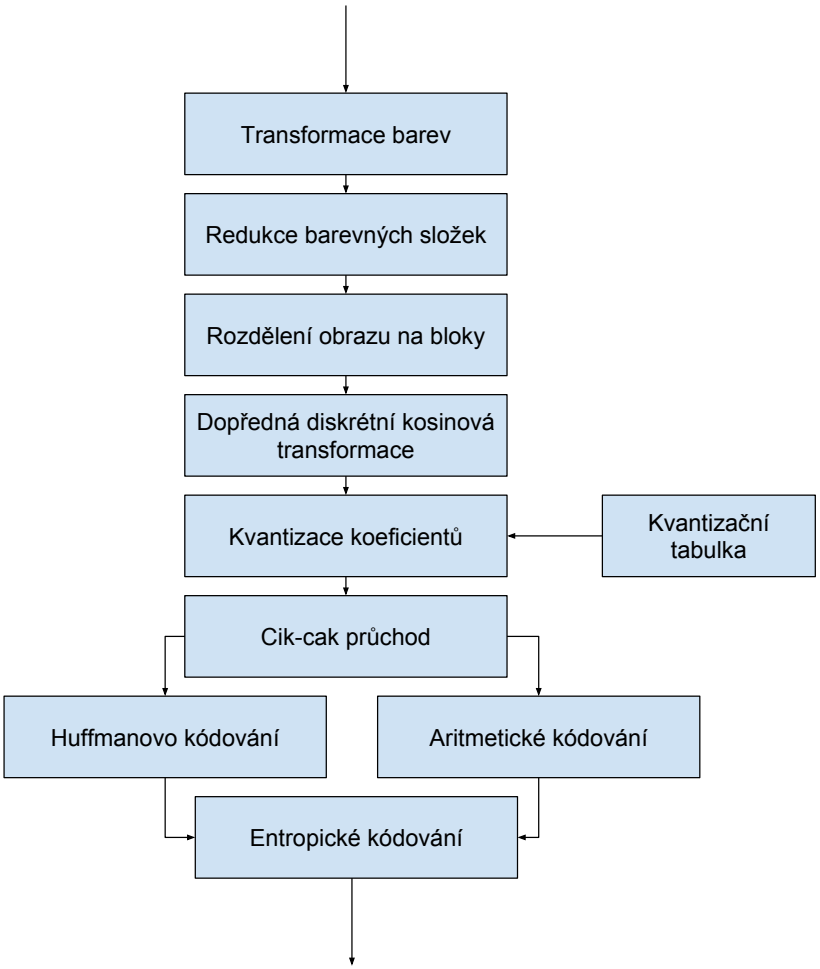
Vstupní signál je u kosinové transformace skládán z průběhů kosinových funkcí o různých amplitudách a frekvencích. Koeficienty u těchto harmonických funkcí nabývají reálných hodnot. Pro výpočet jednorozměrné transformace lze využít vztah (2.9), kde N značí počet vzorků. U výpočtu dvourozměrné transformace se využívá takzvané separabilní transformace, která umožňuje provést za sebou dvě jednorozměrné transformace, a to například nejprve pro řádky a následně pro sloupce bloku obrazu. Vztah (2.10) je určen pro výpočet dvourozměrné kosinové transformace.

$$X_k = \sum_{n=0}^{n-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \tag{2.9}$$

$$X_{k_1,k_2} = \sum_{n_1=0}^{N_1-1}\sum_{n_2=0}^{N_2-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N_1}\left(n_1 + \frac{1}{2}\right)k_1\right]\cos\left[\frac{\pi}{N_2}\left(n_2 + \frac{1}{2}\right)k_2\right] \tag{2.10}$$

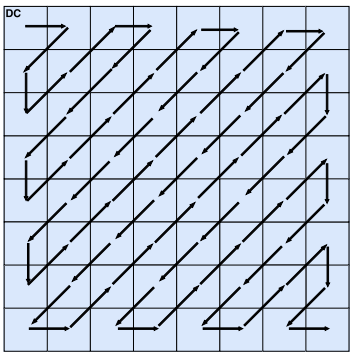
Komprese

Jak komprese probíhá je znázorněno blokovým schématem na obrázku 2.2. Nejdříve se transformují a redukují barevné složky. Dále se vstupní obraz rozloží na bloky obvykle 8x8 pixelů. V dalším kroku se aplikuje na všechny bloky dopředná diskrétní kosinová transformace. Následuje kvantování transformovaných koeficientů kvantizační maticí. V tomto kroku dochází ke ztrátě informace. Posledním krokem JPEG komprese je bezeztrátové kódování kvantovaných koeficientů. Nejdříve se uspořádají koeficienty jednotlivých bloků pomocí Cik-cak průchodu, který je ilustrován na obrázku 2.3.



Obrázek 2.2: Blokový popis komprese obrazu u standardního formátu JPEG.

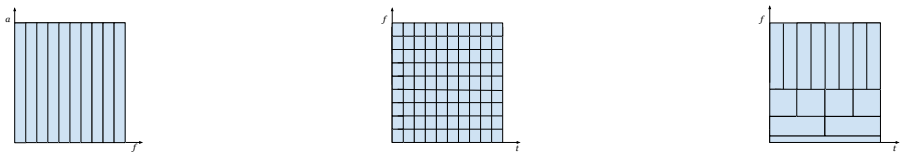
Na začátku průchodu jsou přidány největší absolutní hodnoty a na konci nuly. Samotný bod [0,0] má však jiné statistické rysy než ostatní, protože nemá střední hodnotu rovnu nule. Pro jeho zakódování se navíc ještě využívá takzvané delta kódování. To pracuje tak, že se od něj odečte hodnota koeficientu z předešlého bloku. Pro zakódování nenulových hodnot se využije Huffmanovo nebo aritmetické kódování. U kódování nulových hodnot je využito RLE kódování, kde se posloupnost nul nahradí jejich počtem. Dekomprese probíhá obdobně jako komprese pouze s inverzním postupem.



Obrázek 2.3: Ukázka cik-cak průchodu blokem 8x8.

2.4 Vlnková transformace

Vlnková transformace pro skládání původního signálu nevyužívá oproti kosinové a fourierovy transformaci signály s periodickým průběhem, ale využívá takzvané vlnky, které mohou být hladké, ostré i asymetrické. [13] Fourierova transformace poskytuje především pohled na zastoupení frekvence v signálu. Chybí u ní informace o umístění frekvencí v čase. Krátkodobá fourierova transformace udává zastoupení frekvencí v časovém okně pouze o jeho konstantní velikosti. Zatímco u vlnkové transformace frekvenční i časová okna nemají konstantní velikost, a proto se lépe přizpůsobí původnímu signálu. Pokud zvolené časové okno je široké, dosáhne se tak přesnějšího určení frekvencí, ale zhorší se tím přesnost časové lokality. Naopak výběrem krátkého časového okna dojde ke zlepšení přesnosti detekce časové složky, ale zhorší se frekvenční rozlišení. Tento úkaz nese název Heisenbergův princip neurčitosti a to mezi veličinami času a frekvence u vlnkové transformace. Využívá se toho pro lepší frekvenční rozlišení u nižších frekvencí a vyšší přesnosti lokalizace času u vyšších frekvencí. [6]



Obrázek 2.4: Frekvenční analýza FT, časově frekvenční analýza STFT a časově frekvenční analýzy WT

Rovnicí (2.11)je formálně popsána vlnková transformace, která obecně pracuje se spojitým časem,

$$\gamma(r,s) = \int f(t)\psi^*_{r,s}(t)dt \tag{2.11}$$

kde r, s jsou reálná čísla a $\psi^*_{r,s}$ označuje funkci pro komplexně sdruženou vlnku. Vstupní signál $f(t)$ se převede podobně jako u fourierovy transformace na množiny bázových funkcí.

Rovnice (2.12) znázorňuje výpočet formálního popisu vlnky,

$$\psi_{r,s}(x) = \frac{1}{\sqrt[2]{r}} \cdot \psi \cdot \frac{t-s}{r} \tag{2.12}$$

kde $\frac{1}{\sqrt[2]{r}}$ slouží k normalizaci energie. Tento vztah nedefinuje přesnou funkci pro vlnku, ale umožňuje dosadit parametry pro vlnku vhodnou pro vstupní signál. Vztahem (2.13) je popsán inverzní popis vlnkové transformace,

$$f(t) = \frac{1}{C_\psi} \int_0^{+\infty} \int_{-\infty}^{+\infty} Wf(r,s)\psi_{r,s}(t) ds \frac{dr}{s^2} \tag{2.13}$$

kde s značí posuv a r dilatace. Po diskretizaci těchto parametrů dochází k určení spojité vlnkové funkce.

Vlnka

Vlnková transformace rozkládá vstupní signál pomocí vlnek. Všechny vlnky jsou odvozeny od jedné mateřské vlnky pomocí posunutí a dilatace. Tyto odvozené vlnky se nazývají dceřiné. Množina dceřiných vlnek s mateřskou se označují jako rodina vlnek. Aby se vlnka dala použít jako mateřská musí splňovat tyto podmínky. Vlnka musí mít nulový průměr a jednotkovou plochu. Rovnicí (2.14) lze vypočítat udané podmínky.

$$\int_{-\infty}^{+\infty} \psi(t)dt = 0, \int_{-\infty}^{+\infty} |\psi(t)|^2 dt = 1 \tag{2.14}$$

Další z podmínek je přijatelnost rovnicí definovaná vztahem (2.15).

$$C_\psi = \int_0^{+\infty} \frac{|\hat{\psi}(\omega)|^2}{\omega} d\omega < +\infty \tag{2.15}$$

kde $\hat{\psi}(\omega)$reprezentuje Fourierovu transformaci vlnky ψ. Z rovnic výše potom vyplývá, že výsledek rovnice(2.16) platí pro nulovou frekvenci.

$$|\hat{\psi}(\omega)|^2_{\omega=0} = 0 \tag{2.16}$$

Vlnky splňující tyto podmínky mohou disponovat vlastnostmi ovlivňujícími proces komprese. Jedna z vlastností se nazývá existence kompaktního nosiče. Tuto vlastnost splňuje taková vlnka, která má lokalizovanou svoji energii na konečném časovém úseku. Hlavní výhodou je rychlost výpočtu. Výpočet se provádí konvolucí FIR filtru vlnky se vstupním signálem. V diskrétním čase lze říci, že čím je kratší vlnka tím existuje méně nenulových koeficientů. Tím se snižuje množství výpočetních operací, které se musí vykonat pro zjištění výsledku funkce. Další z vlastností je hladkost vlnky. Hladká vlnka dosahuje při ztrátové kompresi lepších výsledků oproti vlnkám, které nejsou hladké. Dekomprimované hladké vlnky lépe splývají s původním signálem.

Multirozklad

Spojitá vlnková transformace není vhodná pro kompresi obrazu například kvůli tomu, že vstupní signál, který je funkcí jedné proměnné, je transformován na funkci dvou spojitých proměnných. Tyto dvě promněnné se označují dilatace a posunutí. Taková transformace lze

označit jako redundantní. Pro odstranění těchto nadbytečných informací lze využít například dyadické vzorkování a to následovně $r0 = 2$; $s0 = 1$ poté $r = 2^m$; $s = n2^m$ [8]. Dosazení tohoto vzorkování do rovnice vlnky (2.12) a vlnkové transformace (2.11) dostaneme vztah (2.17) pro výpočet Dyadické vlnkové transformace, kde m vyjadřuje frekvenční měřítko a n zastupuje časové posunutí.

$$\gamma(m,n) = \frac{1}{\sqrt[2]{2^m}} \int f(t)\psi^* \left(\frac{t - n2^m}{2^m} \right) dt \tag{2.17}$$

Pomocí multirozkladu lze analyzovat signál $f(t) \in L^2(\mathbb{R})$ po frekvenčních pásmech. Celý tento prostor lze rozložit do podprostoru V_j , který je popsaný rovnicí (2.18).

$$\{0\} \cdots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \cdots L^2(\mathbb{R}) \tag{2.18}$$

Báze podprostoru V_j je pak tvořena $\{\phi_{j,n}\} n \in \mathbb{Z}$ a poté lze jeho ortogonální doplněk W_j popsat vztahem (2.19) a bází tohoto podprostoru odpovídá $\{\psi_{j,n}\} n \in \mathbb{Z}$. Pomocí doplňků rovnicí (2.20)lze vypočítat prostor spojitých funkcí.

$$V_{j-1} = V_j + W_j \tag{2.19}$$

$$L^2(\mathbb{R}) = \cdots \oplus W_2 \oplus W_1 \oplus W_0 \oplus W_{-1} \oplus W_{-2} \oplus \cdots \tag{2.20}$$

Svázáním podprostorů jako je tomu u rovnice (2.19) můžeme vyjádřit dilatačními rovnicemi (2.21) a (2.22), kde $h(n)$ značí FIR filtr s dolní propustí a $g(n)$ zastupuje FIR filtr s horní propustí, které slouží ke svázání ze sousedními podprostory. Pod $\phi(2t - n)$ je v rovnicích zadefinováno podvzorkování. Pomocí těchto rovnic lze vstupní signál rozložit na různé stupně detailu.

$$\phi(t) = \sqrt{2} \sum_n h(n)\phi(2t - n) \tag{2.21}$$

$$\psi(t) = \sqrt{2} \sum_n g(h)\phi(2t - n) \tag{2.22}$$

2.5 Diskrétní vlnková transformace

Již zmiňovaná dyadická vlnková transformce lze označit jako diskrétní vlnková transformace se spojitým časem. Nyní však budeme využívat diskrétní vlnkovou transformaci s diskrétním časem a budeme jí v této diplomové práci označovat DWT. Pomocí vzorce (2.23) dostaneme její výpočet.

$$y_m[n] = \sum_{k=-\infty}^{\infty} x[k]h_m(2^m n - k) \tag{2.23}$$

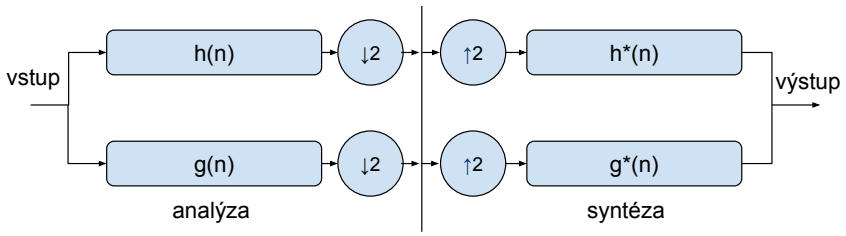
Máme více možností, jak uskutečnit DWT a to buď pomocí konvolučních filtrů a nebo pomocí liftingu. Vstupní signál se filtruje pomocí filtrů s horní a dolní propustí. Horní propust vrací koeficienty s vysokou frekvencí a nízkou energií. Dolní propust naopak, a to tedy s nízkou frekvencí a vysokou energií. Pokud k těmto dvěma filtrům přidáme jejich zrcadlové filtry, dostaneme takzvané kvadraturně zrcadlové filtry. Dále v textu budeme používat zkratku QMF. Blokové schéma QMF lze spatřit na obrázku (2.5).

Biortogonální vlnky

Sada filtrů je určena podle použité vlnky. Nejrozšířenější vlnky pro kompresi obrazu jsou dle [1] biortogonální Cohen-Daubechies-Feauveau, dále v textu je lze najít pod zkratkou CDF. Pro bezeztrátovou kompresi se využívá verze CDF5/3, která je reverzibilní a pro ztrátovou kompresi lze použít typ CDF9/7, který je ireverzibilní. Platí, že čísla v názvech znázorňují velikosti filtrů. Nejdůležitějšími vlastnostmi těchto vlnek je symetrie, která odstraňuje nechtěné artefakty v obraze a velký počet nulových momentů. Čím větší je počet nulových momentů vlnky tím lepší je potom komprese, protože dochází k zahazování nepotřebných informací.

Kaskáda vlnkových filtrů

První úrovní kaskádových vlnkových filtrů je QMF. Opakované použití na výstup dolně propustného filtru předchozí úrovně QMF získáme kaskádový vlnkový filtr vyšší úrovně. Tento postup lze nazvat pyramidový rozklad. Nejvyšší možná úroveň rozkladu signálu až do úrovně jednoho vzorku.

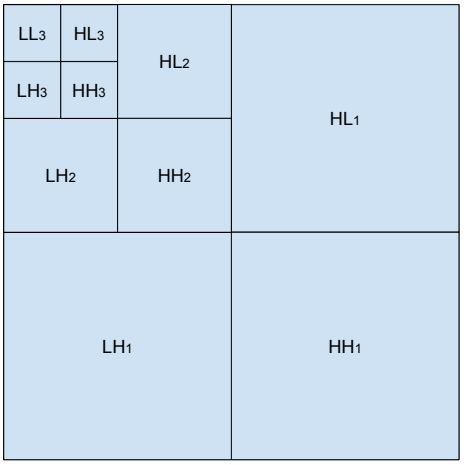


Obrázek 2.5: Blokové schéma kvadraturně zrcadlových filtrů, kde funkce h(n) značí filtr s horní propustí a funkce g(n) označuje filtr s dolní propustí a $\downarrow 2$ značí podvzorkování signálu a $\uparrow 2$ takzvané nadvzorkování sidnálu.

2.6 Diskrétní vlnková transformace ve 2 D prostoru

Doposud jsme psali o diskrétní vlnkové transformaci pro jednorozměrný signál. Obraz je ovšem dvourozměrný (2D), proto si nyní popíšeme, jak se bude postupovat při 2D transformaci. Vstupní obraz se nebude dělit na dvě podpásma jak tomu bylo u 1D signálu, ale rovnou na čtyři jak je tomu na obrázku 2.6. Pro tuto skutečnost jsou zavedeny čtyři filtry (2.24), které jsou kombinací původních dvou filtrů. Tuto transformaci lze nazvat separabilní. Postupuje tak, že se nejdříve provede filtrace řádků a následně se na výstupech provede filtrace sloupců stejně jako u kosinové transformace.

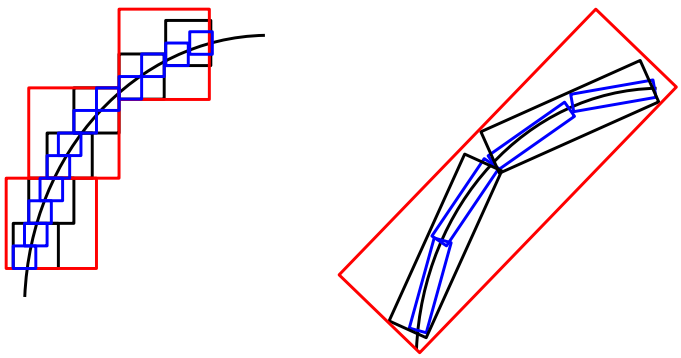
$$\begin{aligned} \phi(n_1, n_2) &= \phi(n_1)\phi(n_2) \\ \psi^H(n_1, n_2) &= \psi(n_1)\phi(n_2) \\ \psi^V(n_1, n_2) &= \phi(n_1)\psi(n_2) \\ \psi^D(n_1, n_2) &= \psi(n_1)\psi(n_2) \end{aligned} \tag{2.24}$$



Obrázek 2.6: Rozklad 2D obrazu na čtyři podpásma do třetího stupně zanoření

2.7 Konturletová transformace

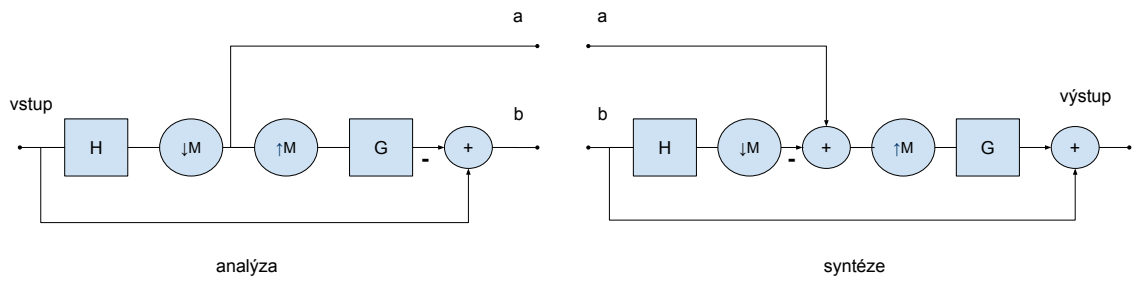
Účinnost reprezentace obrazu závisí na schopnosti zachytit objekty zájmu pomocí co nejmenšího popisu. Vlnky jsou dobré ve 2D při v izolaci nespojitosti v krajních bodech, ale nezprostředkují pohled na hladkost podél obrysů. Kromě toho mohou oddělitelné vlnky zachytit pouze omezenou směrovou informaci. [5] Toto neuspokojivé chování ukazuje, že je zapotřebí silnější zastoupení ve vyšších dimenzích. Proto se používají konturlety, které obsahují kvalitní informace o směrovosti a multirozklad. Na obrázku 2.7 lze vidět ukázku zachycení hran vlnkou a konturletem. Lze pozorovat, že vlnky detekují pouze hrany, zatímco konturlety křivku.



Obrázek 2.7: Ukázka zachycení hran vlnkami a konturlety.

Laplaceova pyramida

Jedním ze způsobů jak získat víceúrovňový rozklad je použití Laplaceovy pyramidy (LP).[5] LP rozklad na každé úrovni generuje podvzorkovanou dolní propust originálního obrazu a rozdíl mezi originálem a predikcí. Obrázek 2.8 znázorňuje proces rozkladu, kde H 2.25 a G2.26 se nazývají (dolnopropustní) filtry pro analýzu a syntézu v tomto pořadí. M je vzorkovací matice. Výstup a představuje hrubou podvzorkovanou verzi původního obrazu. Výstup b znázorňuje rozdíly mezi hrubou a původní verzí. Například aplikací matice M na vstupní signál $x[n]$ získáme podvzorkovaný signál $x_d[n] = x[Mn]$, kde M je celočíselná matice.[5]



Obrázek 2.8: Analýza a syntéza signálu pomocí Laplaceovi pyramidy.

$$\phi(t) = 2 \sum_{n \in \mathbb{Z}^2} g[n]\phi(2t - n) \tag{2.25}$$

$$\psi(t) = 2 \sum_{n \in \mathbb{Z}^2} h[n]\psi(2t - n) \tag{2.26}$$

Směrová banka filtrů

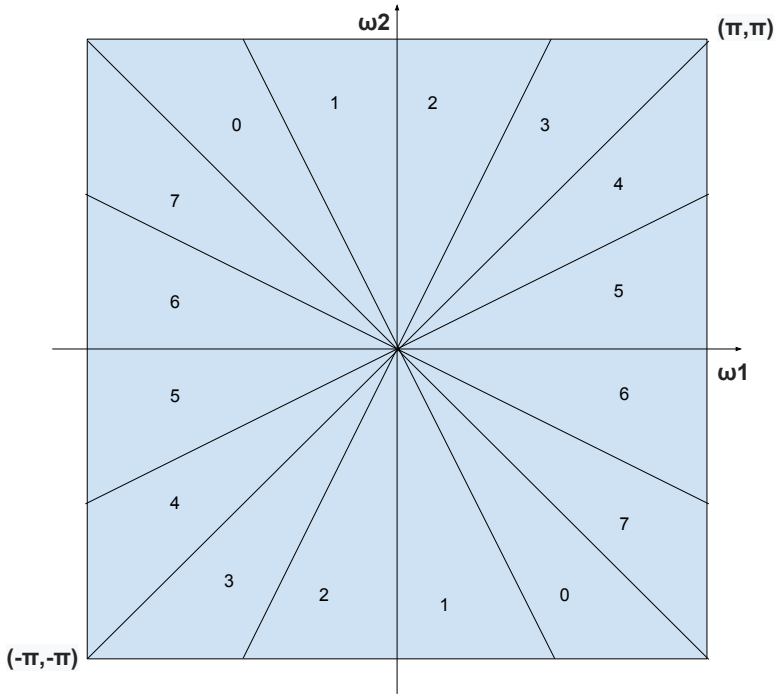
Směrová banka filtrů jak už z názvu vyplývá, slouží k analýze obrazu na rozklad pro jednotlivé směry.[4] Rozklad se vytvoří pomocí binárního stromu úrovně l, ze kterého se vytvoří 2^l podpásem trojúhelníkového tvaru. Náhled rozkladu pro $l = 3$ lze spatřit na obrázku 2.9.

Směrová banka filtrů se skládá ze dvou složek a to dle [14] z *quincunx*, která rozděluje prostor na horizontální a vertikální. Druhou složkou jsou operátory sloužící k natočení původního signálu dle potřeby. Označují se jako *shearing* operátory.

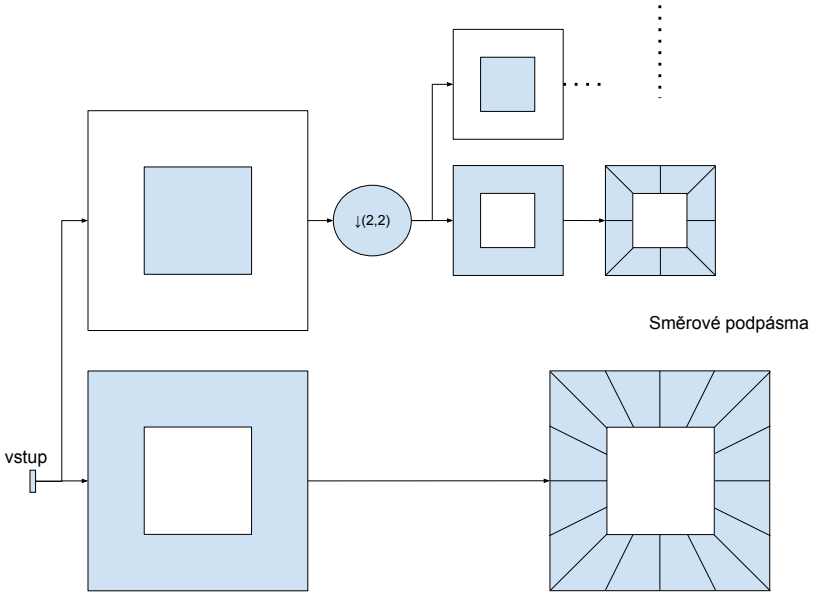
Konturletová banka filtrů

Konturletová banka filtrů kombinuje LP a směrovou banku filtrů pro lepší zachycení objektů nižší frekvence. V podstatě se jedná o aplikaci transformace pro detekci hran a následné lokální směrové transformace pro detekci kontur. Na základě této myšlenky je navržena struktura banky filtrů zobrazena na obrázku 2.10. Pro řídkou expanzi v obrazech s hladkými konturami. [4]

Diskrétní kontruletová transformace splňuje tyto vlastnosti dle [4]:



Obrázek 2.9: Ukázka rozdělení 2D obrazu na osm podpásem.



Obrázek 2.10: Banka filtru typu Contourlet.

1. Pokud se pro LP a směrovou banku filtrů využívají filtry dokonalé rekonstrukce. Pomocí diskrétní konturletová transformace lze vytvořit dokonalý původní obraz.
2. Pokud se pro LP a směrovou banku filtrů využívají filtry ortogonální, pak pomocí diskrétní konturletové transformace lze zajistit těsný rámec ohraničení roven jedné.
3. Horní mez pro poměr redundance diskrétní konturletové transformace je 4/3.
4. Je-li směrová banka filtrů úrovně l_j a j je úroveň pyramidového rozkladu, pak základní obraz konturletové transformace bude mít šířku 2^j a délku 2^{j+l_j-2}.
5. Při využití při výpočtu FIR filtru, bude výpočetní složitost diskrétní kontuletové transformace $\mathbb{O}(N)$ pro obraz, který má N pixelů.

2.8 Kvantování

Algoritmy pro kompresi koeficientů obvykle pracují s celými čísly, ale výstupem jednotlivých transformací jsou reálná čísla. Pro tento převod mezi reálnými a celými čísly se využívá kvantování. Dochází k trvalé ztrátě informací. Přesnost kvantovaných hodnot udává počet a rozptyl hodnot, kterých můžou koeficienty nabývat. Nejsnadnější možnost je zaokrouhlit reálná čísla k nejbližšímu celému číslu. Jedna z dalších možnosti je před samotnou kvantizací vynásobit všechny koeficienty zvolenou konstantou buď pro zvětšení nebo zmenšení rozptylu kvantování.

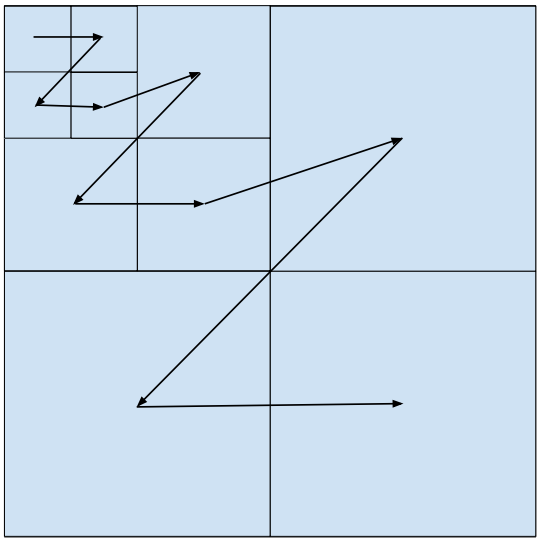
2.9 Kódování koeficientů

V této části práce si přiblížíme algoritmy pro kódování koeficientů DWT, které nejsou sami o sobě ztrátové, kromě zaokrouhlování při výpočtu s reálnými vlnkami u filtrace. Nejvíce ztrátovou operací při kompresi obrazu u formátu JPEG bývá v největší míře kvantování po provedení diskrétní kosinové transformace. [15] U komprese obrazu pomocí DWT už nemusí a obvykle ani není největší ztrátový krok právě kvantování, ale samotné kódování koeficientů, kde nejméně významné informace jsou řazeny až na konec bitového toku. Výsledná komprese závisí až na momentu přerušení toku dat. Obecně lze popsat postup kódování v několika krocích dle [8]:

1. *Inicializace*: uložení indexu n do prvního neprázdné množiny koeficientů Θ_n:
$$n = \lfloor \sup_m \log_2 |f_B[m]| \rfloor \tag{2.27}$$
2. *Kódování významnosti*: uložení mapy významnosti $b_n[m]$ for $m \notin \Theta_{n+1}$.
3. *Zakódování znaménka*: zakódování znaménka koeficientů f_B for $m \in \Theta_n$.
4. *Upřesnění kvantování*: uloží se n-tý bit všech koeficientů splňující podmínku $|f_B[m]| > 2^{n+1}$. Jedná-li se o koeficienty, které patří do určité sady Θ_k pro $k > n$, které už má pozici uloženou. Jejich n-tý bit je uložen v pořadí, v jakém byla jeho pozice zaznamenané v předchozích průchodech.
5. *Zjemnění přesnosti*: snížení n o 1 a pokračování krokem 2.

Nejvýznamnější technikou kódování je skupina využívající hierarchie rozkladu výstupních koeficientů.[16] Základní taková technika se nazývá EZW.[11]

EZW

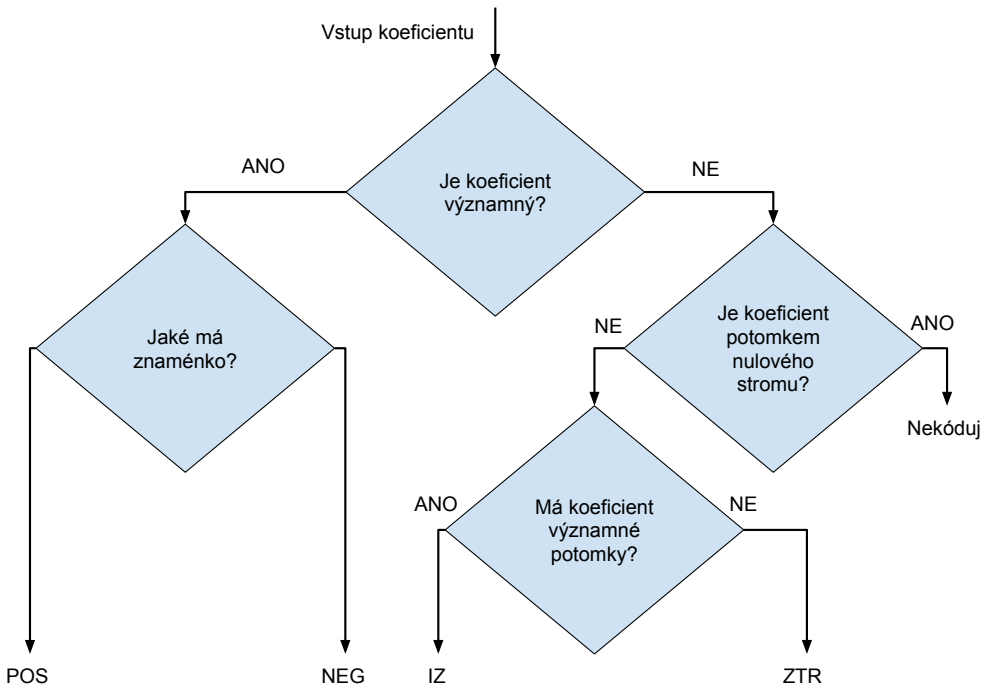


Obrázek 2.11: Průchod EZW mezi třemi úrovněmi koeficientů.

Embeddded zerotree wavelet (EZW) patří mezi nejzákladnější algoritmy pro kódování DWT koeficientů. Stal se předlohou pro mnoho dalších algoritmů například pro Set Partitioning in Hiearchical Trees(SPIHT). Ukázka průchodu EZW mezi třemi úrovněmi koeficientů je vyobrazena v obrázku 2.11. Tento algoritmus patří do třídy progresivního kódování, takže o něm můžeme říci, že čím více dat z výstupu si uchováme tím bude dekomprese přesnější. Jestliže algoritmus dokončí průchod mezi všemi koeficienty bude se jednat o bezeztrátovou kompresi. Pokud jej ukončíme předčasně, tak získáme ztrátovou kompresi. Čím později kompresi ukončíme tím méně informací o obrazu ztratíme. EZW využívá toho, že přírodní obrazy mají většinou více energie v nižších pásmech než ve vyšších. Díky tomu lze říci, že vyšší pásma pouze přidávají detail, a proto je můžeme vypustit. Dále využívá toho, že větší koeficienty jsou významnější než menší. Koeficienty jsou kódovány, jsou-li větší než zvolený práh. Jestliže jsou menší, tak se kódování koeficientů pozdrží dokud nesplní tuto podmínku. Po průchodu všemi koeficienty se zmenší práh a následně dochází k opětovnému dominantnímu průchodu viz obrázek 2.12, kde POS a NEG reprezentuji koeficienty, které se budou kódovat. ZTR označuje kořen nulových stromů, což znamená, že všichni jeho potomci jsou nevýznamní, a proto je nebude potřeba kódovat. IZ v algoritmu představuje koeficient, který je nevýznamný, ale některý z jeho potomků je významný. Výstupem dominantního průchody jsou tedy symboly POS, NEG, ZTR a IZ, na které se následně aplikuje aritmetický kodér. Další část iterace spočívá v zakódování koeficientů dle mapy významnosti vytvořené z předchozích kroků. Iterace se opakují dokud nedojde k překročení maximálního bitového toku nebo nebudou zakódovány všechny koeficienty.

SPIHT

Set Partitioning in Hierarchical Trees (SPIHT) vychází z EZW, který vylepšuje. Jeho embedded kódování je označováno jako *progresive transmition scheme*. Podobně jako EZW kóduje nejdříve nejvíce významné koeficienty, protože z nich lze vypozorovat největší část



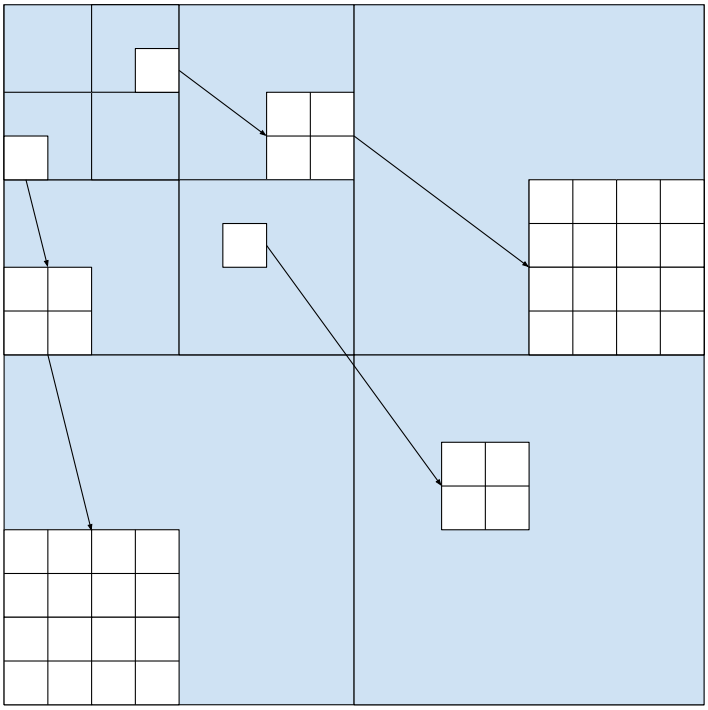
Obrázek 2.12: Dominantní průchod EZW pro tvorbu mapy významnosti

informace o obraze. Algoritmus při svém průběhu bere v úvahu spojitost mezi koeficienty na různých úrovních rozkladu. Systém kódování je různý od EZW, protože informace o pořadí výstupních bitů není explicitně přenášena. Spoléhá na to, že pokud je algoritmus pro kodér i dekodér stejný, může se využít *executition path* algoritmus. Ten je definován rozhodnutím na každém větvení algoritmu. Dekodér za pomoci těchto rozhodnutí může zjistit původní cestu a pomocí ní vydedukuje, jak byly bity seřazeny při kódování.[10] Výhodou při absenci informace o pořadí bitového výstupu dochází k tomu, že nejsou zapotřebí symboly jako tomu bylo u EZW a tím pádem není obvykle potřeba následného aritmetického kódování.[9] Výstup algoritmu je čistě bitový.

Prostorové stromy

SPIHT pracuje s takzvanými prostorovými stromy (*spatial orientation trees*). Tato datová struktura je vytvořena předešlou transformací například DWT. Strom má počátek v kořenu. V případě, že je signál jednorozměrný bude mít každý kořen dva potomky. U obrazu, který je dvourozměrný, bude mít kořen čtyři následovníky. Jedná se konkrétně o prostorovou korelaci jednotlivých koeficientů mezi úrovněmi rozkladu. Příklad rozkladu na obrázku 2.13. Algoritmus SPIHT dle [10] definuje tyto množiny koeficientů:

1. $\mathcal{O}(i,j)$: množina souřadnic všech přímých potomků uzlu (i,j)
2. $\mathcal{D}(i,j)$: množina souřadnic veškerých podřízených uzlů uzlu (i,j)
3. $\mathcal{H}(i,j)$: množina souřadnic kořenů všech prostorových stromů



Obrázek 2.13: Prostorový strom vzniklý vlnkovou transformací.

4. $\mathcal{L}(i,j)$: množina souřadnic všech nepřímých potomků uzlu

$$\mathcal{L}(i,j) = \mathcal{D}(i,j) - \mathcal{O}(i,j)$$

Algoritmus

Algoritmus pro realizaci využívá tři typy seznamů a to seznam nevýznamných množin *LIS*, seznam významných množin *LIP* a seznam významných bodů *LSP*. Seznamy *LIP* a *LSP* představují jednotlivé koeficienty. Seznam *LIS* je dvojího typu. První typ se označuje A a představuje $\mathcal{D}(i,j)$. Druhý typ je označován B a představuje $\mathcal{L}(i,j)$.

Algoritmus lze popsat dle [10] do čtyř kroků:

1. *Inicializace*: nastavení počátečních hodnot seznamů a kvantizačního kroku (1)
2. *Srovnávací průchod*:(2)
   - průchod seznamem *LIP*, kde jsou koeficienty testovány na významnost a případně přesunuty do LSP
   - průchod seznamem *LIS*, kde jsou koeficienty opět testovány na významnost testovány podle uvedených pravidel případně rozděleny.
3. *Upřesňovací průchod*: průchod položkami seznamu LSP, kromě těch, co byly přidány v poslední srovnávacím průchodu a následné odeslání nejvýznamnějšího bitu podle aktuálního kvantizačního kroku. (3)
4. *Aktualizace kvantizačního kroku*: snížení kvantizačního kroku o 1 a následně pokračuje krokem 2.(4)

Algoritmus 1 SPIHT - Inicializace

$n := \lfloor \log_2(MAX); \rfloor$, $LSP := \varphi$, $LIP := \mathcal{H}$
$LIS :=$ pouze ty prvky z $\mathcal{H}$, které mají přímé potomky jako prvky typu A;

Algoritmus 2 SPIHT - Srovnávací funkce

for each$(i,j) \in LIP$ **do**
    odešli $S_n(i,j)$;
    if $S_n(i,j) = 1$ **then**
        přesuň (i,j) do LSP a odešli znaménko $c_{i,j}$;
for each$(i,j) \in LIS$ **do**
    (Zpracování prvků typu A)
    if prvek je typu A **then**
        odešli $S_n(\mathcal{D}(i,j))$;
        if $S_n(\mathcal{D}(i,j)) = 1$ **then**
            for each$(k,l) \in \mathcal{O}(i,j)$ **do**
                odešli $S_n(k,l)$;
                if $S_n(k,l) = 1$ **then**
                    přidej (k,l) do LSP a odešli znaménko $c_{k,l}$;
                if $S_n(k,l) = 0$ **then**
                    přidej (k,l) na konec LIP;
            if $\mathcal{L}(i,j) \neg\varphi$ **then**
                přesuň (i,j) na konec LIS jako prvek typu B;
                go to (Zpracování prvku typu B);
            else
                odeber z LIS prvek (i,j);
    (Zpracování prvků typu B)
    if prvek je typu B **then**
        odešli $S_n(\mathcal{L}(i,j))$;
        if $S_n(\mathcal{L}(i,j)) = 1$ **then**
            přidej každý $(k,l) \in \mathcal{O}(i,j)$ na konec LIS jako prvek typu A;
            odeber (i,j) z LIS;

Algoritmus 3 SPIHT - Upřesňovací průchod

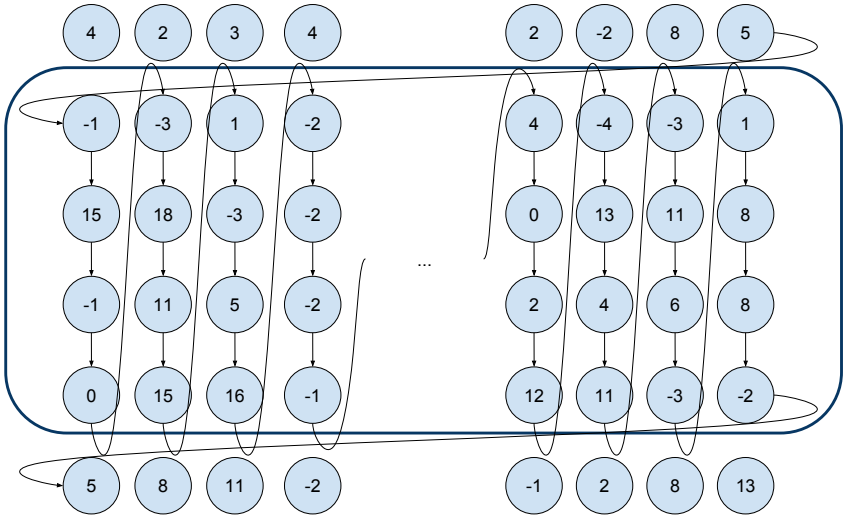
for each$(i,j) \in LSP$ kromě těch prvků, které byly přidány v posledním srovnávacím průchodu **do**
    odešli n-tý nejvýznamnější bit $|c_{i,j}|$;

Algoritmus 4 SPIHT - Aktualizace kvantizačního kroku

$n := n - 1$;
go to (Srovnávací průchod);

EBCOT

Embedded Block Coding with Optimized Truncation je jednou z možností jak pro kódování DWT koeficientů. Snaží se nevytvářet závislosti mezi jednotlivými úrovněmi rozkladu a tím pádem chyba ovlivní pouze danou úroveň. Tento algoritmus je použit ve formátu JPEG 2000. Algoritmus nejprve rozdělí DWT koeficienty obvykle na bloky o velikosti 64.[3] Všechny bloky jsou na sobě nezávislé. U každého bloku se provede průchod, který spočívá v rozdělení bloku na řádky o velikosti čtyři a jednotlivé průchody se provedou po sloupcích. Přechod mezi řádkovými částmi bloků se provedou tak, že poslední koeficient předchozího pruhu postupuje na první koeficient aktuálního pruhu.



Obrázek 2.14: Průchod koeficientů v rámci jednoho bloku EBCOT

Princip kódování

EBCOT je založen na principu iterativního kódování bitových rovin. Na každou takovou rovinu je třeba aplikovat tři průchody pro vyhodnocení a to Significance propagation, Magnitude refinement a Cleanup pass. Ty v jednotlivých průchodech přiřazují hodnoty využívaným primitivům, které reprezentují tabulku obsahující kontexty daného koeficientu z ostatních podpásem stejného bloku. Primitiva jsou označovány jako Run-length, Zero coding, magnitude refinement a Sign coding. Takový to průchod je znázorněn na obrázku 2.14. Zakódované informace se dále zakódují pomocí MQ kodéru.

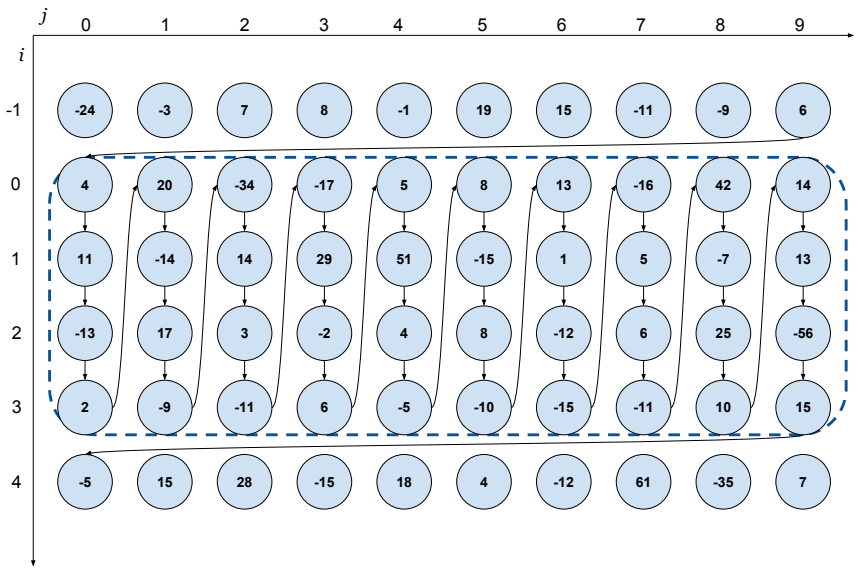
Significance propagation – V tomto průchodu se kóduje pouze osmiokolí významného koeficientu. Je to z důvodu toho, že důležité koeficienty se vyskytují ve shlucích. V průchodu se využívají pro kódování pouze primitiva Zero coding a Sign coding.

Magnitude refinement – U tohoto průchodu dochází ke zpřesnění, které koeficienty jsou správě v předchozích průchodech označovány za významné a které nikoliv. Využité primitivum je pouze magnitude refinement.

CleanUp pass – Tento průchod zpracovává a kóduje ty koeficienty, které nebyly kódovány pomocí předešlých průchodů. Využívá primitiva Run-length, Zero coding a Sign coding. Dále pracuje s unikátním symbolem UNI, který slouží k identifikaci prvního významného koeficientu ve sloupci. Tato pozice musí být jednoznačně určena. Dokud se nenalezne významný koeficient ve sloupci, tak jsou všechny dosavadní sloupce zakódovány jedním Run-length primitivem. Pokud se nalezne významný bit, tak je zakódováno znaménko koeficientu pomocí Sign-conding a zbývající hodnoty ve sloupci jako Zero-coding.

Příklad EBCOT

Protože algoritmus EBCOT je stěžejním algoritmem celé práce ukážeme si postup kódování na příkladu převzatém z [2]. Předpokládejme kódování bloku znázorněného na obrázku 2.15, který se nachází v sub-pásmu DWT označovaného LH. Všechny hodnoty v buňkách znázorňují DWT koeficienty po kvantizaci. Nejvýznamnějším koeficientem na obrázku 2.15 je na pozici $i = 4aj = 7$, a to tedy s hodnotou 61. Tento blok tedy bude mít 6 úrovní bitových hladin ($2^6 = 64$). Budeme je značit od 0 do 5, kde 5 představuje nejvýznamnější bitovou hladinu.



Obrázek 2.15: Část kódovaného bloku využitého u příkladu.

Kódování nejvýznamnější bitové roviny

Na této bitové hladině znázorněné na obrázku (2.16) bude pouze proveden *Cleanup Pass*. Ostatní dva průchody se neprovedou, protože neexistují významné koeficienty získané z vyšší úrovně. Na obrázku 2.16 znázorňují černé body významné koeficienty v této bitové rovině.

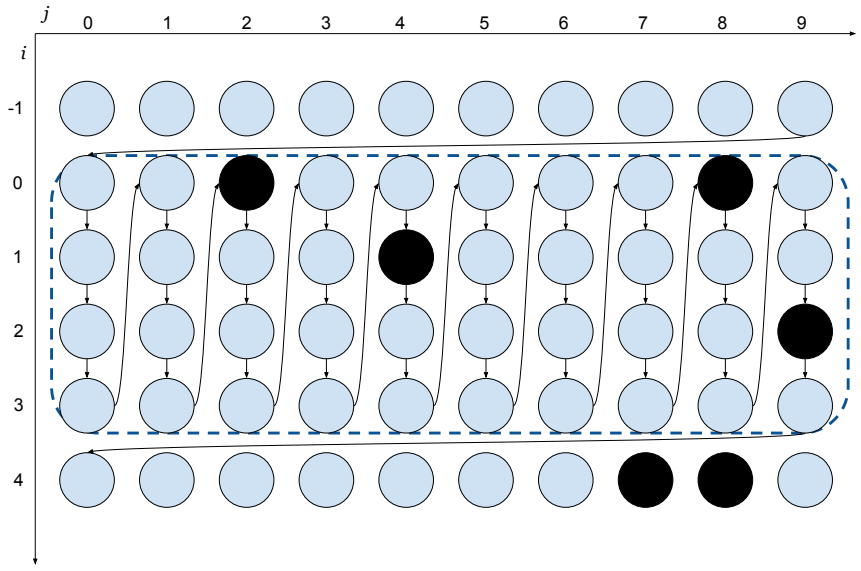
Průchod *Cleanup Pass*

Průchod začne prvním sloupce (j=0). V tomto sloupci nejsou žádné významné koeficienty a není znám žádný významný koeficient v sousedství, proto bude na výstup *Run-Length*(RL) zaslán symbol 0, který značí v této bitové rovině sloupec nul (nevýznamných koeficientů). V druhém sloupci (j=1) se opět nenachází významný bit, proto se opět na výstup RL pošle symbol 0. V třetím sloupci (j=2) zůstává kodér v režimu RL, dokud nenarazí na významný koeficient nebo dokud se nenajde významný koeficient v sousedství. Hned první koeficient (i=0, j=2) je významný, proto se ukončí režim RL zasláním symbolu 1 na výstup. Navíc průchod indikuje první významný koeficient ve zpracovávaném sloupci, který se označí symbolem 00 (UNI). Tento symbol je zaslán na výstup, a dále pomocí něj dekodér určí přesnou

pozici, kde byl ukončen RL režim. Znaménko koeficientu se kóduje jakmile je koeficient označen za významný v dané bitové rovině. Pro kódování znamének využívat EBCOT predikci založenou na znaménkách čtyř sousedů. Tyto predikce jsou uvedeny v tabulce 2.1 a to pro sub-pásma LL, LH a HH. Hodnota $\overline{x}^h$ udává znaménka dvou horizontálních sousedů a hodnota $\overline{x}^v$ pak znaménka dvou vertikálních sousedů. V případě HL jsou veličiny inverzní. Hodnota $\overline{x}$ se rovná 1 tehdy, když jsou oba sousedé kladní nebo jeden je kladný a druhý ještě nemá znaménko. Hodnota $\overline{x}$ se rovná 0 tehdy, pokud oba sousedé nemají znaménko nebo znaménka sousedů jsou opačná. Hodnota $\overline{x}$ se rovná -1 tehdy, pokud jsou záporní oba sousedé nebo pokud jeden ze sousedů nemá znaménko a druhý je záporný. Pokud je predikce $\hat{x}$ korektní, výstupem je symbol 0 jinak 1.

$\overline{x}^h$	$\overline{x}^v$	K^{SC}	$\hat{x}$
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

Tabulka 2.1: Predikce znaménka a *Sign coding*.



Obrázek 2.16: Bitová hladina n = 5.

Predikce koeficientu (i=0,j=2) je kladná a koeficient je záporný, proto je na výstup (SC0) poslán symbol 1. Pro zbývající koeficienty v třetím sloupci (j=2) již nelze využít RL, tak se na koeficienty použije *Zero coding* ZC. Užívá se devět různých označení od ZC0 do ZC8. Jsou založeny na známé významnosti osmi okolních sousedů. Jsou definovány v tabulce 2.2. Hodnoty k^h, k^v a k^d představují hodnoty horizontálních, vertikálních a diagonálních sousedů, kteří již byli označeny za významné.

LL a LH			HL			HH		k^{ZC}
k^h	k^v	k^d	k^h	k^v	k^d	k^d	$k^h + k^v$	
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

Tabulka 2.2: Princip kódování Zero Coding (ZC). „x“ značí jakoukoliv hodnotu.

Koeficient (i=1, j=2) má pouze jednoho významného souseda a to nad sebou. Navíc není významný v této bitové rovině. Na výstupu (ZC3) je tedy symbol 0. Ostatní koeficienty ve sloupci (j=2) nemají významného souseda. Výstupem zde jsou dva symboly nula (ZC0). V čtvrtém sloupci (j=3) je pro kódování užito ZC. Koeficient (i=0, j=2) je významný. Zakódování všech koeficientů ve sloupci bude 0(ZC5), 0(ZC1), 0(ZC0) a 0(ZC0). V pátém sloupci (j=4) je opět možné povolit režim RL, protože není znám významný koeficient ani soused. Koeficient (i=1, j=4) je prohlášen za významný, proto bude zapsán na výstup (RL) symbol 1. Na výstup se také zapíše symbol 01, jelikož se jedná o první významný koeficient ve sloupci. Znaménko je predikováno jako SCO a je kladné, takže na výstup (SC0) je zapsán symbol 0. Další dva koeficienty kódovány pomocí ZC. Výstup je potom 0(ZC3) a 0(ZC0). Zbývající kódování je popsáno v tabulce 2.3 pro bitovou rovinu n=5.

Kódování bitové roviny n=4

Druhá bitová rovina je znázorněna na obrázku 2.17. Významné koeficienty s předchozí roviny jsou vyobrazeny černou barvou. Šedou barvou jsou označeny koeficienty, které byly shledány významnými v aktuální rovině. Nejprve se provede *Significance Propagation* a potom *Magnitude Refinement*.

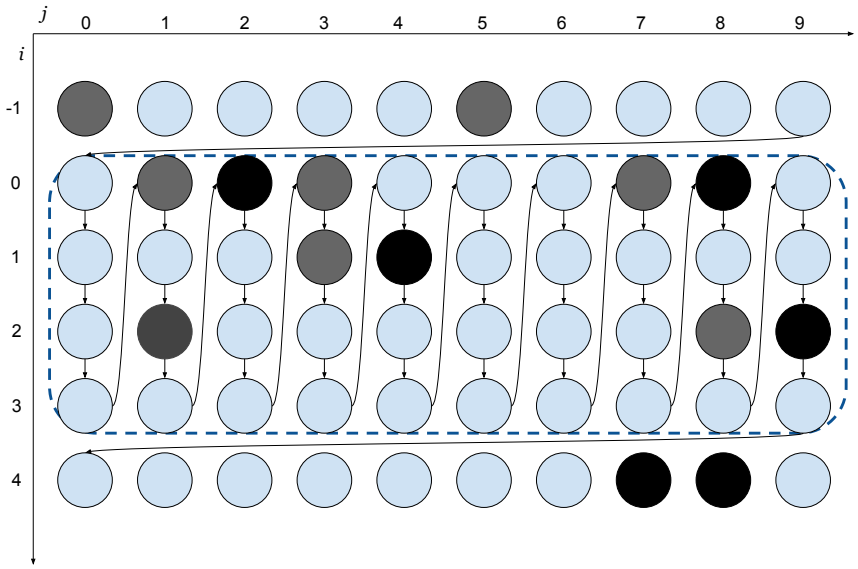
Průchod *Significance Propagation*

Počáteční koeficient (i=0, j=0) má souseda (i=-1, j=0), který se stal v této rovině významným. Bit na pozici (i=0, j=0) je tedy kódován SP. V tomto průchodu se využívá ZC. Bit (i=0, j=0) se zakóduje na výstup (ZC3) posláním symbolu 0. Je známo, že zbylé koeficienty ve sloupci nemají významného souseda a proto se nebudou kódovat SP a jsou označeny křížkem v obrázku 2.18.

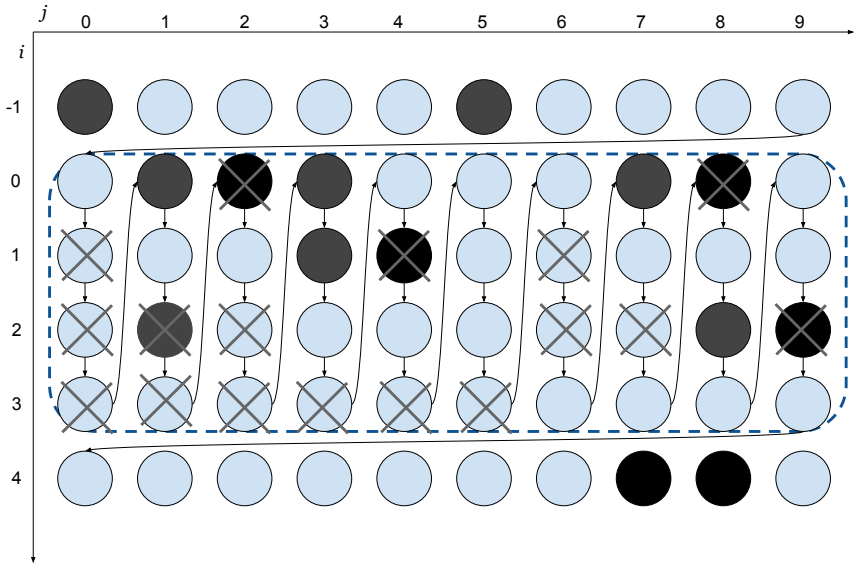
Koeficient na pozici (i=0, j=2) je významný, proto zakódujeme koeficienty (i=0, j=1) a (i=1, j=1). Bit (i=0, j=1) je v aktuální rovině významný, proto je na výstup (ZC6) poslán symbol 1. Znaménko je zakódováno posláním na výstup (SC3) symbol 1, protože predikce nesouhlasí. Pro zakódování koeficientu (i=1, j=1) se pošle na výstup (ZC3) symbol

	Kódované symboly	Komentář
j=0	0(RL)	RL
j=1	0(RL)	RL
j=2	1(Rl) 00(UNI) 1(SC0) 0(ZC3) 0(ZC0) 0(ZC0)	RL do koeficientu (i=0, j=2) Pozice prvního významného koeficientu. Predikce znaménka v souvislosti (SC0) není korektní. Zbytek sloupce ZC.
j=3	0(ZC5) 0(ZC1) 0(ZC0) 0(ZC0)	RC není aktivní (pokračuje ZC) Koeficient (i=0, j=2) je významný.
j=4	1(RL) 01(UNI) 0(SC0) 0(ZC3) 0(ZC0)	RL do koeficientu (i=1, j=4) Pozice prvního významného koeficientu. Predikce znaménka v souvislosti (SC0) je korektní. Zbytek sloupce ZC.
j=5	0(ZC1) 0(ZC5) 0(ZC1) 0(ZC0)	RC není aktivní (pokračuje ZC) Koeficient (i=1, j=4) je významný.
j=6	0(RL)	RL
j=7	0(RL)	RL
j=8	1(RL) 00(UNI) 0(SC0) 0(ZC3) 0(ZC0) 0(ZC0)	RL do koeficientu (i=0, j=8) Pozice prvního významného koeficientu. Predikce znaménka v souvislosti (SC0) je korektní. Zbytek sloupce ZC.
j=9	0(ZC5) 0(ZC1) 1(ZC0) 1(SC0) 0(ZC3)	RC není aktivní (pokračuje ZC). Koeficient (i=0, j=8) je významný. Významný koeficient v souvislosti (SC0) není korektní. Predikce znaménka v souvislosti (SC0) není korektní. Zbytek sloupce ZC.

Tabulka 2.3: Kódování bitové hladiny na úrovni 5 pomocí průchodu Cleanup Pass



Obrázek 2.17: Bitová hladina n = 4.



Obrázek 2.18: *Significance Propagation* v bitové hladině n = 4.

0. Ostatní koeficienty ve sloupci nejsou kódovány. První koeficient ve třetím sloupci (j=2) není kódován, protože byl kódován v předešlé rovině. Bit (i=1, j=2) zakódujeme posláním na výstup (ZC3) symbol 0. Ostatní koeficienty ve sloupci nejsou kódovány. Zbývající kódování SP v tabulce 2.4.

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

Tabulka 2.4: Kódování bitové hladiny na úrovni 4 pomocí průchodu Significance Propagation

Průchod *Magnitude Refinement*

Kóduje bity, které byly významné v předešlé bitové hladině. Má tři typy kódování MR0, MR1 a MR2 a jsou popsány v tabulce 2.5. Pokud je $\tilde{o} = 0$, tak byl použit poprvé MR na kódovaný koeficient. Jinak je hodnota $\tilde{o} = 1$. Veličiny k^h a k^v mají stejný význam jako u ZC. Hodnoty koeficientů, které byly v předchozí bitové rovině jsou zpřesňovány. Symbol 0(MR1) je poslán v případě zpřesnění koeficientu (i=0, j=2). 0(MR1) 1(MR1) jsou poslány na výstup k upřesnění koeficientů (i=1, j=4), (i=0, j=8) a (i=2, j=9).

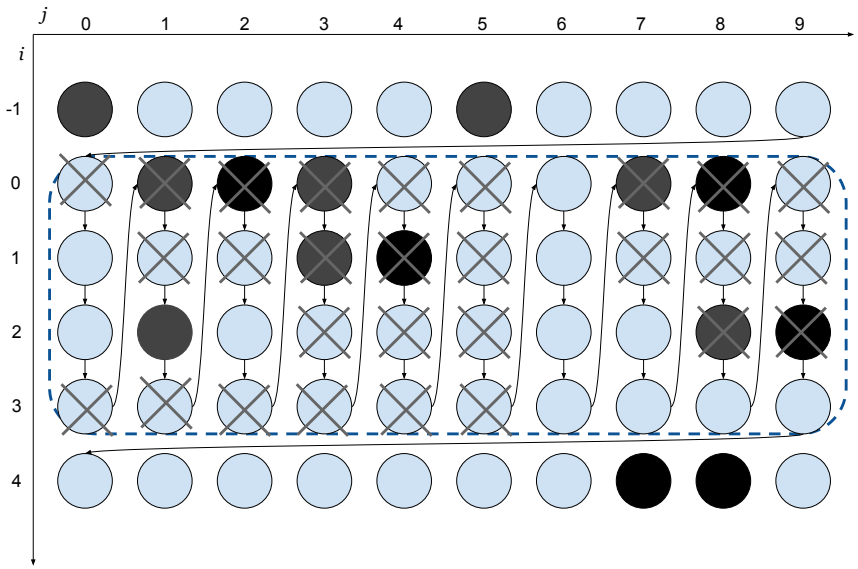
$\tilde{\{o\}}$	$k^h + k^v$	k^{MR}
0	0	MR0
0	$\neq 0$	MR1
1	x	MR3

Tabulka 2.5: Typy znaků kódování MR

Průchod *Cleanup Pass*

Tento průchod zakóduje bity, které nebyly zakódovány v předchozích průchodech v rovině a jsou na obrázku 2.19 označeny křížkem. Kódování popsáno v tabulce 2.6.

Tento postup se opakuje až do bitové hladiny n=0. Tento kódovací proces je první částí samotného EBCOT kodéru zvaný Tier1.



Obrázek 2.19: Cleanup Pass v bitové hladině n = 4.

	Kódované symboly	Komentář
j=0	0(ZC1) 0(ZC0) 0(ZCO)	
j=1	1(ZC0) 0(SC0) 0(ZC3)	Zakódován koeficient (i=2, j=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)	
j=8	0(ZC3) 1(ZC5) 1(SC3) 0(ZC4)	
j=9	0(ZC5) 0(ZC3) 0(ZC3)	

Tabulka 2.6: Kódování bitové hladiny na úrovni 4 pomocí průchodu Cleanup Pass

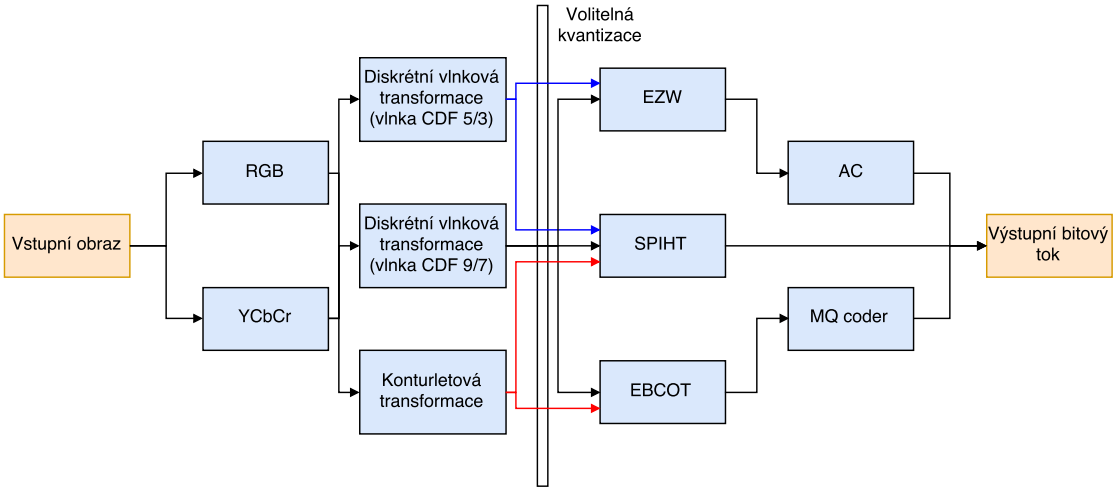
Kapitola 3

Implementace

Tato část práce, jak už nadpis napovídá, se bude zabývat dílčím částem implementace kompresní knihovny. V podkapitole návrh knihovny je implementace rozčleněna do několika oddílů kompresního řetězce. Dále jsou tyto moduly popsány jak jsou konkrétně implementovány ve výsledné aplikaci.

3.1 Návrh knihovny

Dílčím úkolem celé práce bylo vytvořit knihovnu pro srovnání kvality komprese a dekomprese obrazových dat dle zvolených specifikovaných parametrů jako je typ barevného prostoru, typ transformace a algoritmu pro zakódování transformovaných dat. Celý návrh knihovny je zaznamenán v blokovém schématu na obrázku 3.1.



Obrázek 3.1: Návrh kompresní knihovny.

Protože knihovna není orientována na rychlost komprese, ale je především určena pro porovnání kvality výstupů zvolených komprimačních řetězců, tak je možno implementaci rozčlenit do aplikačních modulů.

3.2 Vstupy a výstupy

Samotná knihovna je určena pouze jako nástroj pro kompresi a dekompresi obrazových dat. Pro jednoduší ovládání knihovny byla vytvořena aplikace, která dle vstupních parametrů modifikuje nastavení knihovny pro danou kompresi a následně srovná kvalitu komprese.

Pro porovnání kvality komprese byly implementovány v knihovně funkce s metrikami PSNR a SSIM.

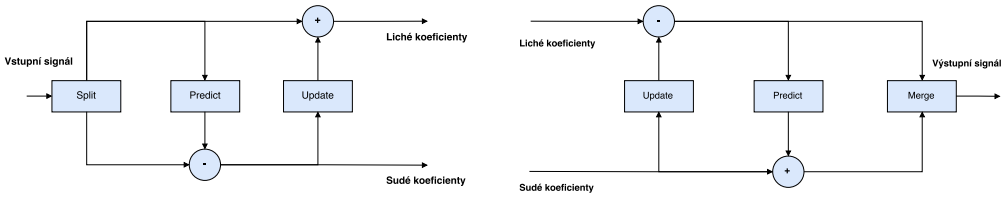
3.3 Realizace

Implementace knihovny probíhala v jazyce C++ pod operačním systému *Microsoft Windows 10*. Pro psaní zdrojových textů a překlad knihovny bylo využito vývojové prostředí *Microsoft Visual Studio 2015*.

Pro tvorbu grafů v této technické zprávě byl využit program *gnuplot* verze 5. Pro kreslení blokových schémat byl využit nástroj *draw.io*, který pracuje ve webovém prostředí. Ostatní obrázky kromě těch, které vytvořila samotná aplikace, byly nakresleny jako *Nákresy Google*.

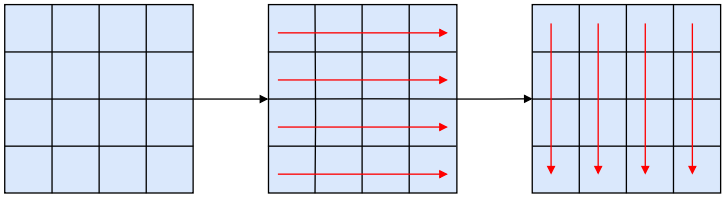
Implementace diskrétní vlnkové transformace

Výpočet diskrétní vlnkové transformace je v knihovně uskutečněn pomocí techniky `lifting`. Hlavní výhodou této techniky je bezesporu menší časová složitost než u uskutečnění výpočtu transformace pomocí konvoluce. Blokové schéma dopředné a zpětné transformace pomocí liftingu je znázorněn na obrázku 3.2. Na obou obrázcích lze zjistit symetrii obou postupů. Inverzní transformace má pouze opačně znaménka a obráceně aplikuje úkony `predict` a `update`.



Obrázek 3.2: Blokové schéma dopředné a zpětné transformace pomocí lifting techniky pro 1D signálu.

Aby se tato technika dala aplikovat pro 2D signál(obrazová data), je možno využít separability, pomocí které se transformuje dvourozměrný signál na dva po sobě jdoucí jednorozměrné signály a to pro sloupce a řádky.



Obrázek 3.3: Postup separabilní transformace obrazu.

Diskrétní vlnková transformace v knihovně

Pro vykonání vlnkové transformace v knihovně je nutné vytvořit objekt dle třídy `CDF97` (nebo `CDF53`), která obsahuje celou dopřednou tak i zpětnou transformaci vlnky pro ztrátovou kompresi CDF 9/7 (nebo pro bezeztrátovou kompresi CDF 5/3)s koeficienty z tabulky 3.1 převzaty z [8]. Pro aplikaci dopředné transformace je potřeba zavolat metodu objektu `forward2d`. Vstupem teto funkce je obraz ve formátu `cv::Mat` a číslo úrovně zanoření. Nejdříve funkce rozloží vstupní obraz na matice dle jednotlivých barevných složek, pokud se ovšem nejedná o šedotónový obraz. Dále pro jednotlivé úrovně zanoření transformace se nejdříve pro řádky a následně i pro sloupce vyvolá funkce pro výpočet jednorozměrné diskrétní vlnkové transformace `forward`. Tato funkce na jednorozměrná data u již zmí-

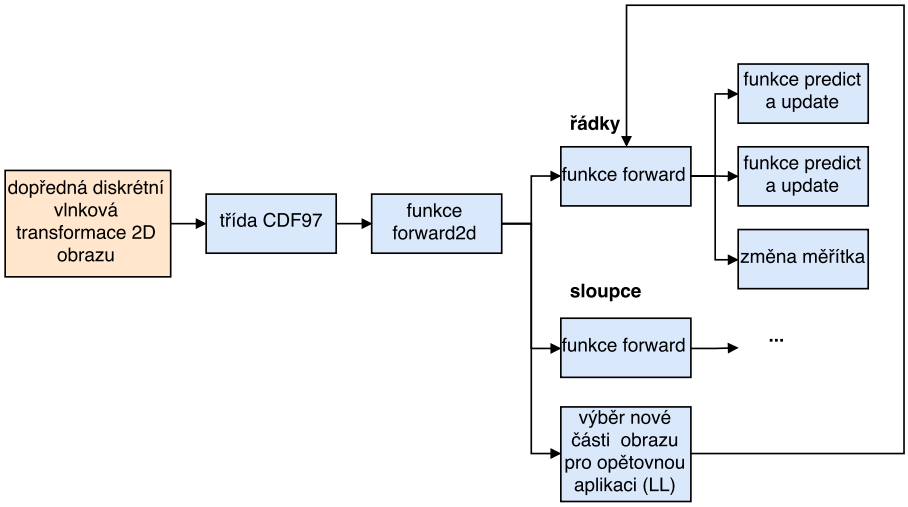
něné vlnky aplikuje postupně funkce `predict`, `update`, `predict` a následně `update`. Dále se aplikuje postup nazvaný `scale`, který rozdělí vstupní data na důležitá a méně důležitá.

	CDF 9/7	CDF 5/3
Predict_1	1.58613434342059	0.5
Update_1	-0.0529801185729	0.25
Predict_2	-0.8829110755309	
Update_2	0.4435068520439	
Scale_1	1.1496043988602	1.41421356237309504880
Scale_2	1/1.1496043988602	1/1.41421356237309504880

Tabulka 3.1: Koeficienty využité při implementaci

Výstupem transformace, se kterým se dále pracuje při kódování koeficientů, je vektor objektů třídy `DwtImage`. Tyto objekty nesou informace o tom, o jaký blok se jedná, jaká je úroveň transformace, o jakou jde barevnou složku a nakonec celý transformovaný obraz bloku ve formátu `cv::Mat`.

Pro vykonání opačné tedy inverzní transformace lze využít funkce `inverse2d`. Vstupem této funkce je dvourozměrný obraz, kde nejvíce významné bloky se nachází v levém horním rohu. Postupně se od nejvýznamnějších čtyř částí obrazu aplikuje funkce `inverse`, nejdříve pro sloupce a následně pro řádky, kde nakonec dochází ke sloučení informací o obraze. Metoda `inverse` nejprve aplikuje opačný postup `scale` a nasleduje postupné volání funkcí `update`, `predict`, `update`, `predict` s opačnými znaménky než u dopředné transformace. Tento celý postup se aplikuje pro všechny úrovně transformace.

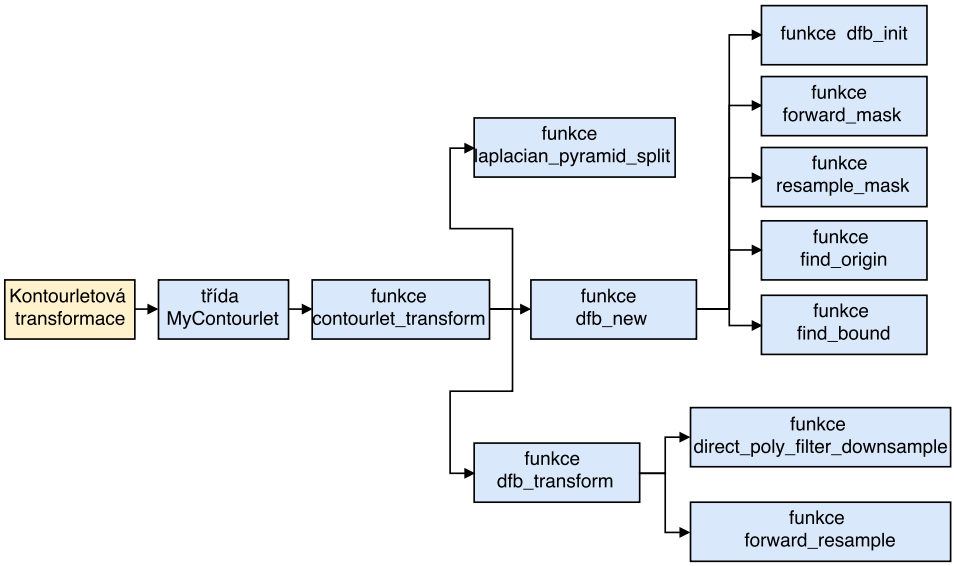


Obrázek 3.4: Posloupnost volání funkcí u dopředné diskrétní vlnkové transformace.

Implementace konturletové transformace

Při implementaci konturletové transformace byla využita již vytvořená implementace v jazyce C ze zdroje [**?**]. Tato implementace nepoužívá diskrétní vlnkovou transformaci pro rozklad obrazu.

Nejdříve je nutné vytvořit novou strukturu `contourlet_t` a pomocí funkce `contourlet_new` se vyvolá její inicializace. Samotná realizace kotourletové transformace je uskutečněna pomocí funkce `contourlet_transform`. Tato funkce má na vstupu inicializovanou strukturu `contourlet_t` a načtenou matici ve formátu `mat`. Celý postup transformace se skládá z několika dílčích kroků. V prvním kroku je aktuální aproximační obraz rozdělen na nízkofrekvenční a vysokofrekvenční obrazy pomocí laplaceovy pyramidy. Dále jsou vysokofrekvenční snímky podrobeny filtrováním podle filtru `dfb_new`. Směrový rozklad se následně provede funkcí `dfb_transform`. Zpětná transformace je implementována v souboru `contourlet.cpp` pod názvem `contourlet_itransform`.



Obrázek 3.5: Posloupnost volání funkcí u dopředné contourletové transformace.

Rozklad pomocí Laplaceovi pyramidy je implementován ve funkci `laplacian_pyramid_split`. Ten má za úkol rozdělit vstupní obraz do dvou pásem Laplace a to nízkofrekvenčních a vysokofrekvenčních. Namísto toho sloučení se provádí funkcí `laplacian_pyramid_merge`, která obnovuje původní obraz.

Pro vytvoření banky filtrů je zapotřebí vyvolat funkci `dfb_new` a pro transformaci `dfb_transform`. Nakonec pro implementaci je využito makro `FILTER`. Zpětná transformace banky filtrů se uskuteční funkcí s názvem `dfb_itransform`.

Funkce `dfb_new` potřebuje jako vstupní parametry velikost vysokofrekvenčního obrazu, který se má zpracovávat a hodnotu úrovně rozkladu. Návratový typ této funkce je struktura `dfb_t`. Inicializace symetrie filtrů se uskuteční funkcí označenou `dfb_init`. Směrový rozklad se provádí přechodem mezi kořeny a listy celého binárního stromu. Pro každý objekt stromu se inicializuje rotační a vzorkovací matice a následně se alokuje prostor pro

nové následovníky. Pokud je objekt list dochází k převzorkování výstupních matic. Pro všechny potomky se s využitím funkce `forward_mask` vypočtou jejich masky. A pokud se znovu jedná o list stromu, je potřeba na vytvořené masky aplikovat převzorkování tentokrát funkcí `resample_mask` a následně zjistit počáteční a hraniční pozici v daných maskách funkcemi `find_origin` a `find_bound`.

Funkce `dfb_transform` nejprve nastaví inicializační informace o koeficientech do binárního stromu sloužícího k rozkladu. Pro celý tento strom se po úrovních vypočítá jeho rozklad s využitím funkce `direct_poly_filter_downsample` a pokud se jedná o jeho poslední úroveň, tak dojde k převzorkování vytvořených dat pomocí funkce `forward_resample`. Nakonec jsou zjištěná data s poslední úrovně zkopírovány do cílového pole.

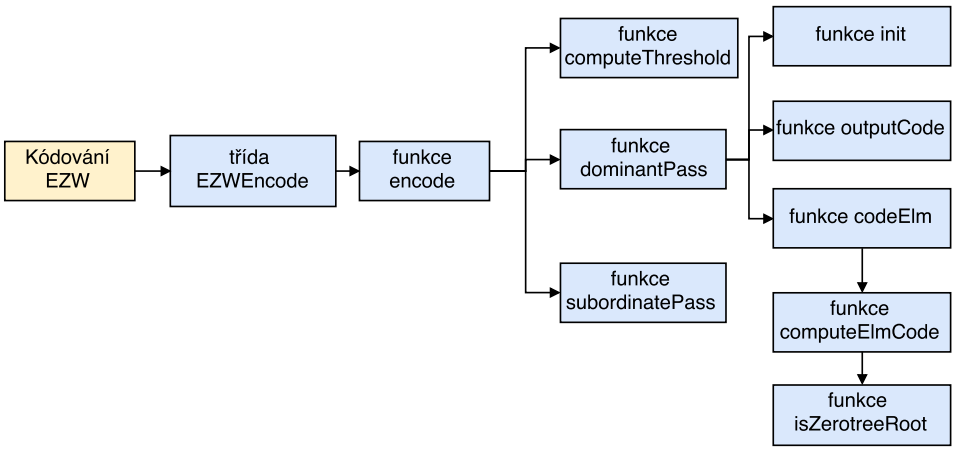
Zpětná konturletová transformace se provede obdobně jako dopředná s tím rozdílem, že se jednotlivé transformační funkce budou vykonávat v opačném pořadí a bude využito jejich inverzní znění(`dfb_itransform`, `backward_resample`, `direct_poly_interpolate_upsample`).

Implementace kvantování koeficientů

Pro snížení počtu kódovaných bitových rovin je možné v knihovně využít skalární kvantování s fixním krokem. Postupuje se tak, že podělí koeficienty velikostí kvantizačního kroku a následně jsou zaokrouhlena na celá čísla. Velikost kvantizačního kroku se mění v závislosti na úrovni rozkladu obrazu. Pro získání bezeztrátové komprese je vždy tato hodnota rovna jedné.

Implementace kódování EZW

Základní kostra implementace kódování EZW vychází z popisu v teoretické části. Pro implementaci dopředného kódování byla vytvořena třída `EZWEncode`. Tato třída obsahuje metodu `encode`, která nejdříve zjistí práh vstupního obrazu s využitím funkce `computeThreshold`. Následně dokud nebude vypočtený práh menší než určený minimální bude postupně aplikovat funkce `dominantPass` a `subordinatePass`. Prvně jmenovaná funkce nejprve inicializuje objekt pomocí funkce `init`. Ta probíhá tak, že na první čtyři body z obrazu aplikuje funkci `codeElm`. Zjištěná hodnota pro bod s pozice (0,0) se rovnou zakóduje s využitím funkce `outputCode`. Ostatní výsledky se uloží do fronty pro další zpracování. Následně se v podmíněném cyklu prochází prvky této fronty dokud nebude prázdná a to tak, že se vybere první prvek fronty a zakóduje se jeho kód pomocí `outputCode`. Zjistí se jestli se nejedná o symbol pro nulový strom a pokud ne, vypočtou se pro něj minimální a maximální ohraničující pozice. Pro tyto body uvnitř dochází pomocí funkce `codeElm` zjištění nových hodnot, které se vloží na konec fronty pro další zpracování.



Obrázek 3.6: Posloupnost volání funkcí u kódování EZW.

Funkce `codeElm` získá z parametrů funkce celý obrazový vstup ve formátu `Matrix`, pozici nového prvku v hodnotách souřadnic x a y a nakonec aktuální práh. Jako první se vytvoří nový prvek se vstupními souřadnicemi. Dále se zjistí aplikací funkce `computeElmCode` prvku jeho nový symbol. Symbol může nabývat čtyř hodnot a to `Pos`, `Neg`, `ZeroTreeRoot` nebo `IsolatedZero`. Pokud je výsledný symbol `Pos` nebo `Neg`, tak se jeho absolutní hodnota z obrazu uloží do listu pro zpracování funkcí `subordinatePass` a následně se jeho hodnota v obraze nastaví na nulu. Návratová hodnota funkce `codeElem` je nově vytvořený prvek.

Jak již bylo popsáno, funkce `computeElmCode` přiděluje prvkům jeden ze čtyř symbolů dle následujících podmínek. Pokud je absolutní hodnota koeficientu menší než aktuální práh a jedná se o nulový strom, pak symbolizuje nový prvek symbol `ZeroTreeRoot`. V případě, že u předešlého předpokladu není splněna podmínka o nulovém stromu, pak se nastaví

hodnota symbolu na `IsolatedZero`. Jestliže nebude absolutní hodnota koeficientu menší než aktuální práh a zároveň hodnota koeficientu nabývá nezáporné hodnoty, přiřadí se k elementu symbol `Pos`. Nesplňuje-li prvek ani jednu z předešlých podmínek, bude označen `Neg`.

V podřadném průchodu ve funkci `subordinatePass` se prochází všechny prvky listu, naplněného hodnotami koeficientů z funkce `codeElem`. Na hodnotu ze seznamu a hodnotu prahu se aplikuje logický *and* a pokud je výsledek nenulový, tak se do bitového výstupu uloží hodnota 1 jinak 0.

Implementace kódování SPIHT

Základem pro implementace kódování SPIHT tvořil popis algoritmu z teoretické části, který byl dále upraven a rozšířen pro lepší správu koeficientů, vytvořených jak vlnokovou tak i konturletovou transformací. Pro zakódování se v knihovně využívá třída `SPIHT_Encoder` a využitím její metody `encode` se spustí samotné kódování. Vstupem metody je ukazatel na výstupní bitový tok ve formátu `BitOutputStream`, který je definovaný hodnotou s maximální počtem bitů zakódování. Prvním krokem kódování je inicializace seznamů `LIP`, `LIS` a `LSP`. Následně se zjistí velikost kroku pro maximální bod v obraze funkcí `initialize`. Dále se v cyklu provádí postupně srovnávací průchod a upřesňovací průchod a poté zmenšování kroku, dokud nebude krok nulový nebo velikost výstupního bitového toku nepřekročí určenou mez.

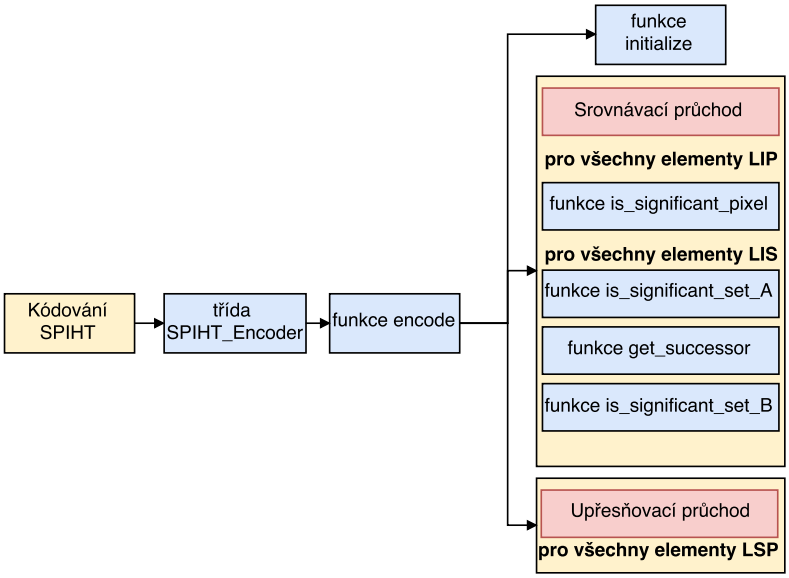
Ve srovnávacím průchodu se nejprve projdou všechny elementy seznamu `LIP` a zjistí se s využitím funkce `is_significant_pixel` zda hodnota bodu v obraze aktuálního kroku je významná. Tato hodnota se vloží do bitového toku. Pokud bude významnost potvrzena daný prvek se vloží do seznamu `LSP`, odstraní se z `LIP` a do bitového toku vloží hodnotu nula, jestliže je hodnota prvku v obraze větší než nula jinak se uloží hodnota jedna. Dále se v tomto průchodu projdou všechny prvky seznamu `LIS`. Nejdříve se zjistí typ elementu.

U prvku typu A se zavolá funkce `is_significant_set_A` pro zjištění významnosti a výsledek se vloží na konec bitového toku. Jestliže byl element označen za významný, tak jsou zpracováni přímý potomci zpracovávaného prvku. Zjištění přímých potomků lze docílit zavoláním funkce `get_successor`. Každý významný potomek je přiřazen do seznamu `LSP` a navíc je uloženo jeho znaménko na bitový výstup. Nevýznamný potomek je pouze uložen do seznamu `LIP`. Posléze se zjistí zda oplývá aktuální prvek seznamu nepřímými potomky. Pokud se potvrdí předešlá podmínka, je prvek vložen do seznamu `LIS` jako element typu B. Nakonec se testovaný prvek ze seznamu `LIs` odstraní.

U prvku typu B ze seznamu `LIS` se zavolá funkce `is_significant_set_B` pro zjištění významnosti a výsledné znaménko se vloží na výstup. Zda-li bude element označen za významný, tak se všichni jeho nepřímý potomci vloží do seznamu `LIS` jako nové prvky typu `A`. Závěrem se zpracovávaný prvek odstraní ze seznamu `LIS`.

Upřesňovací průchod zpracovává v cyklu prvky ze seznamu `LSP`. Jednotlivé absolutní hodnoty prvků v obraze porovná s prahem a pokud je práh menší než zjištěná hodnota, dojde k uložení bitové hodnoty obrazu na pozici aktuálního prahu do bitového toku.

Inverzní kódování je implementováno ve třídě `SPIHT_Decoder` a je prováděno obdobně jako kódování s tím rozdílem, že nejsou potřeba funkce pro zjišťování významnosti, protože tyto hodnoty se čtou přímo ze vstupního bitového toku.



Obrázek 3.7: Posloupnost volání funkcí u kódování SPIHT.

Implementace kódování EBCOT

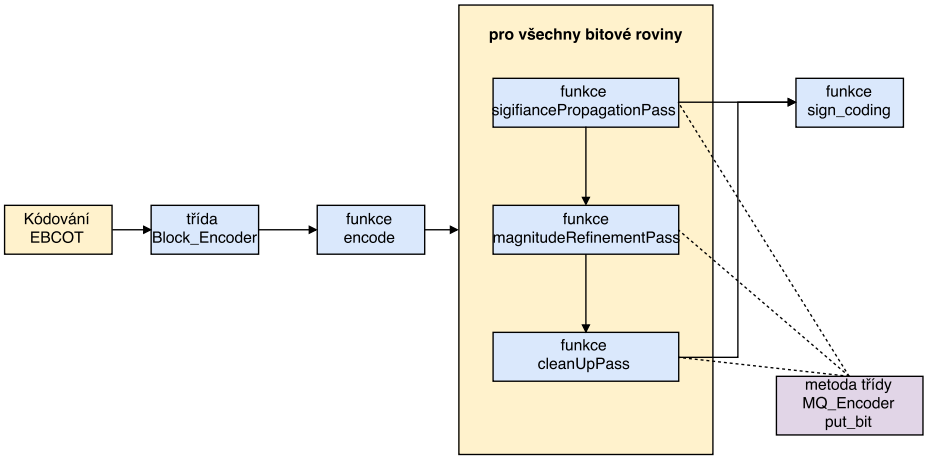
Implementace algoritmu EBCOT vychází z jeho patentu [12]. EBCOT v knihovně se orientuje pouze na kódování označované dle standardu `Tier1` nikoliv na následnou reorganizaci bitového toku.

Postup

Nejdříve se vytvoří objekt dle třídy `Block_Encoder`, která provádí kódování jednotlivých bloků předešlého transformačního rozkladu. Následně se pro blok zjistí počet bitových rovin, ve kterých figurují jeho hodnoty. Toto zjištění i se samotným blokem se předají jako parametry u volání metody `encode`. Uvnitř této funkce se resetuje objekt, který slouží pro kódování hodnot vytvořený dle třídy `MQ_Encoder`. Kódování začíná od nejvíce významné roviny až po tu nejméně významnou. U první zpracovávané roviny lze začít ihned průchodem takzvaným úklidovým, který je implementovaný funkcí `cleanUpPass`, protože neexistují významné bity získané z předešlé roviny. Pokud má vstupní blok více jak jednu bitvou rovinu, jsou dále tyto roviny procházeny v cyklu postupně všemi třemi průchody.

Průchod `sigifiancePropagationPass`, který má na starosti zda má být bit významný v aktuální bitové rovině a je v knihovně zpracován stejnojmennou funkcí, postupuje takto:

- V cyklu se prochází čtyřřádkové části vstupního bloku.
- U každého koeficientu v dané bitové rovině se zjišťuje, zda je dle vzorku sousedů významný.
- Pokud byl významný dle sousedů, dojde k nulovému zakódování.



Obrázek 3.8: Posloupnost volání funkcí u kódování EBCOT.

- Následně se zjišťuje, zda nebyl samotný bit významný, pokud byl pokračuje se jeho zakódováním funkcí `sign_coding`.

Po průchodu určeného pro zjištění významnosti je zavolána funkce `magnitudeRefinementPass`, která slouží ke zpřesnění zakódování již zjištěných významných koeficientů, která pracuje takto:

- Opět se v cyklu prochází vstupní blok po čtyřech řádcích.
- U jednotlivého koeficientu bloku se odhalí, zda je významný.
- V případě, že je koeficient významný z předešlých bitových rovin a zároveň nebyl kódován v průchodu významnosti, potom se postupuje dle následujících kroků:
  - Nejprve se zjistí, zda nebyl koeficient již jednou zakódován v tomto průchodu, pokud ano zakóduje se stavový symbolem k tomu určeným.
  - Zdali ještě nebyl koeficient zakódován v tomto průchodu a jeho okolí je významné, zakóduje se další specifikovanou stavovou hodnotou.
  - Jestliže není ani okolí koeficientu významné zakóduje se jeho bit dle třetí stanovené hodnoty u tohoto průchodu.
- Jinak se postupuje na další koeficient.

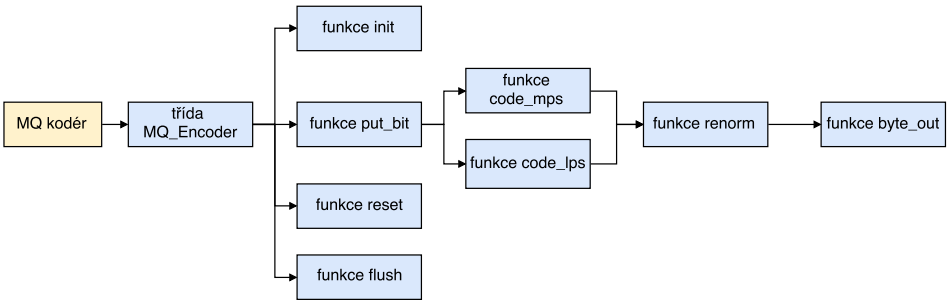
Jako poslední v průchod bitovou rovinou bloku je zavolána funkce `cleanUpPass`, která funguje takto:

- Znovu stejně jako tomu bylo v předešlých průchodech se v cyklu prochází čtyřřádkové části vstupního bloku.
- Pokud byl koeficient kódován některým z předešlých průchodů, pokračuje se dalším v pořadí.
- Jestliže nebyl koeficient ještě kódován a jeho vertikální souřadnice je násobkem čtyř, je zapotřebí vykonat tento postup:

  - Pokud vzorek významnosti všech čtyř koeficientů ve sloupci v dané bitové rovině je nevýznamný a zároveň samotné hodnoty nejsou významné dochází k takzvanému `RunLengthCoding` pro zakódování více stejných symbolů.
  - Pakliže není splněna předešlá podmínka dochází k jeho nulovému zakódování a následně pokud je bit nenulové hodnoty je na něj aplikována funkce `sign_coding`
- Jinak následuje nulové zakódování a pokud je samotný bit významný tak i znaménkové zakódování.

Implementace MQ kodéru

Nejdříve se vytvoří objekt dle třídy `MQ_Encoder`. Následně se u objektu inicializuje pravděpodobnostní tabulka, která výrazně zrychluje kódování. Dále se pro všechny bity a jejich kontexty získané kódováním aplikuje funkce `put_bit`, která zjistí zda bit odpovídá kontextové hodnotě tabulky pro MPS, pokud ano zavolá se pro vstupní kontext funkce `code_mps` pro zakódování MPS, jinak se kontext kóduje funkcí `code_lps` pro zakódování LPS.



Obrázek 3.9: Posloupnost volání funkcí MQ kodéru.

Kódování MPS:

- Vypočte se nová hodnota kódu a nová horní mez intervalu.
- Pokud nová mez nepřekročila maximální, nastaví se nová hodnota MPS dle tabulky.
- Následně se zdvojnásobí horní mez intervalu spolu s kódem funkcí `renorm` do té doby, dokud bude mez menší nebo rovno maximální.

Kódování LPS:

- Vypočte se nová horní mez intervalu.
- Nastaví se nová hodnota LPS dle tabulky.
- Následně se zdvojnásobí horní mez intervalu spolu s kódem funkcí `renorm` do té doby, dokud bude mez menší nebo rovno maximální.

Pokud se v průběhu funkce `renorm` počítadlo bitů navýší na osm bitů je zavolána funkce `byte_out`, která uloží vznikly kód do výstupního pole bajtů.

Kapitola 4

Srovnání

Pro srovnání kvality komprese byly využity funkce algoritmů PSNR a SSIM, které jsou implementovány v knihovně. Výsledky jednotlivých srovnání algoritmů a postupů komprese jsou znázorněny ve spojnicových grafech pro jednotlivé metriky ku velikosti komprimovaného souboru. Následně jsou vizuálně porovnány výřezy komprimovaných obrázků o stejné velikosti.

Pro porovnání knihovny s formátem JPEG2000 byla využita jeho implementace v programu Kakadu 7.9, kde se nastavovala výsledná kvalita parametrem `-rate`.

4.1 Testovací vzorky

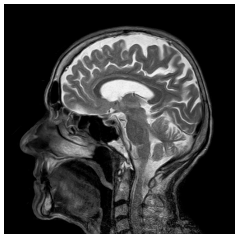
Pro testování byly vybrány tři barevné fotografie a tři šedotónové obrázky, které jsou vhodnější pro porovnání transformací. Jedná se o obrázky o velikosti 512x512 pixelů. Obrázek 4.1a s názvem *Lena* byl zvolen, protože je hojně využíván v pracích zabývajícími se kompresí obrazu.

(a) Lena

(b) Fruits

(c) Watch

(d) Fingerprint1



(e) Xray

(f) Fingerprint2

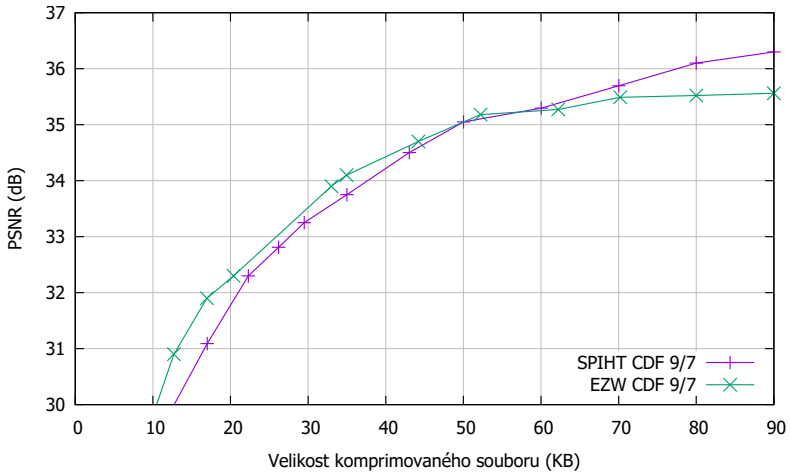
Obrázek 4.1: Testovací vzorky.

4.2 Srovnání dle metrik pro určení kvality

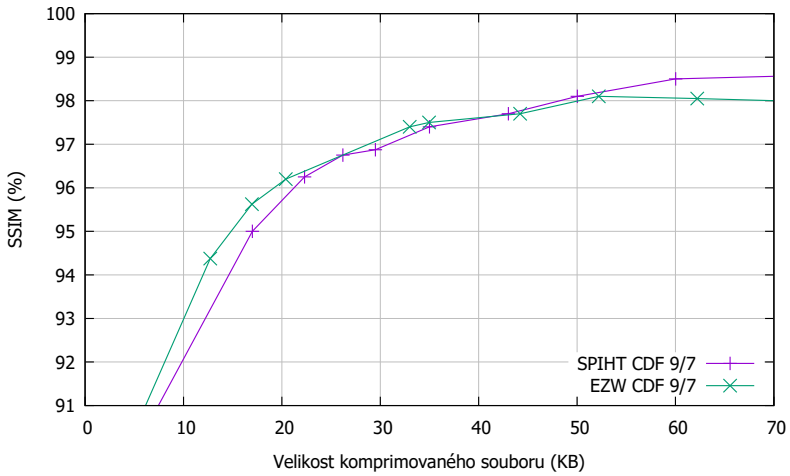
V této podkapitole dojde ke srovnání kvality komprese s využitím porovnávacích metrik.

Srovnání komprese kódování EZW a SPIHT

Jak již bylo zmíněno v teoretické části algoritmus SPIHT vychází z EZW algoritmu. Tím pádem je velká pravděpodobnost, že SPIHT bude efektivnějším algoritmem. Jak je vidět na grafech 4.2, 4.3, počáteční fáze obou algoritmů je do značné míry podobná, jakmile však překročí velikost komprimovaného souboru přibližně padesát kilobajtů, začne mít algoritmus SPIHT strmější průběh(tedy lepší). Co se týká výpočetní náročnosti jednotlivých algoritmů, potřebuje algoritmus SPIHT o poznání delší čas než algoritmus EZW.



Obrázek 4.2: PSNR: Srovnání algoritmů EZW a SPIHT v průměru pro šestici testovaných vzorků.

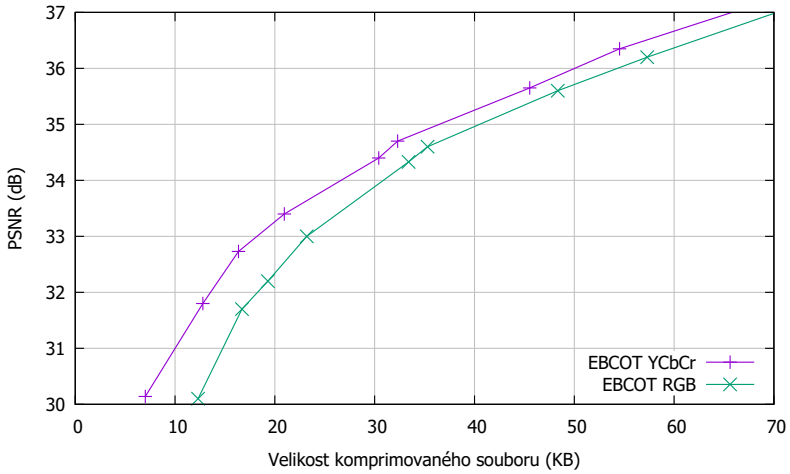


Obrázek 4.3: SSIM: Srovnání algoritmů EZW a SPIHT v průměru pro šestici testovaných vzorků.

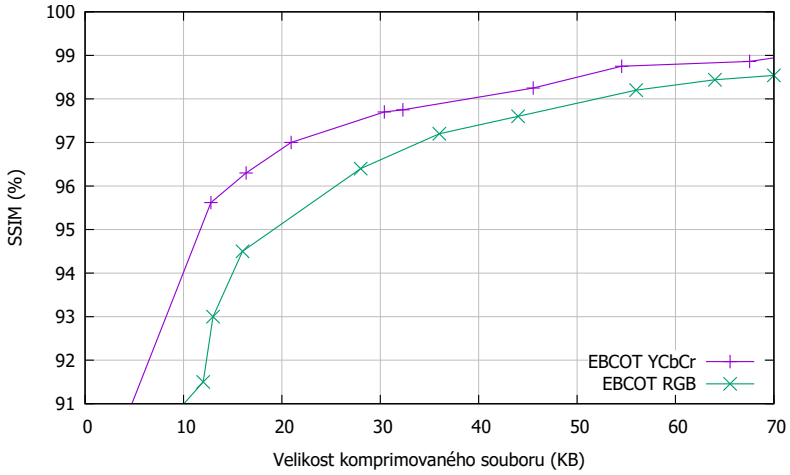
V tomto srovnání, jak se dalo předpokládat, zvítězil algoritmus SPIHT.

Srovnání komprese dle volby barevného prostoru

Knihovna podporuje šedotónový barevný prostor a dva barevné prostory. První z nich se nazývá RGB a druhý ireverzibilní YCBCR, který se hojně využívá ve ztrátových kodecích. Nyní srovnáme kvalitu komprese obou barevných modelů. Pro porovnání bylo zvoleno kódování koeficientů EBCOT a DWT s vlnkou CDF9/7 a první trojice testovacích obrázků.



Obrázek 4.4: PSNR: Srovnání kvality komprese dle barevného prostoru v průměru pro trojici barevných testovaných obrázků.

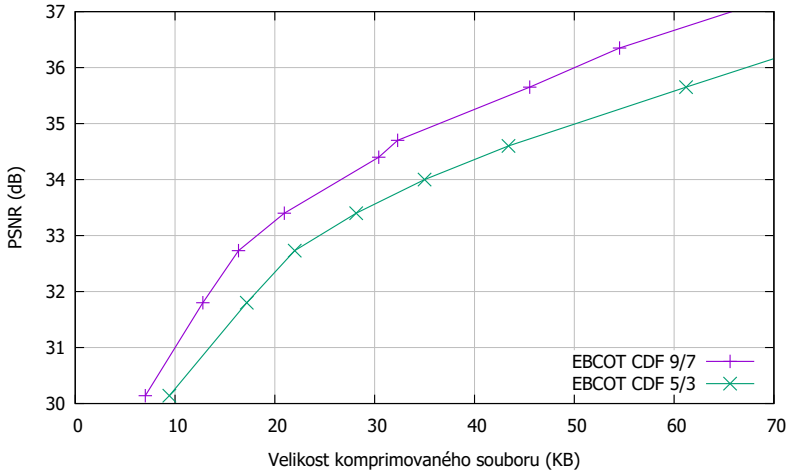


Obrázek 4.5: SSIM: Srovnání kvality komprese dle barevného prostoruv v průměru pro trojici barevných testovaných obrázků.

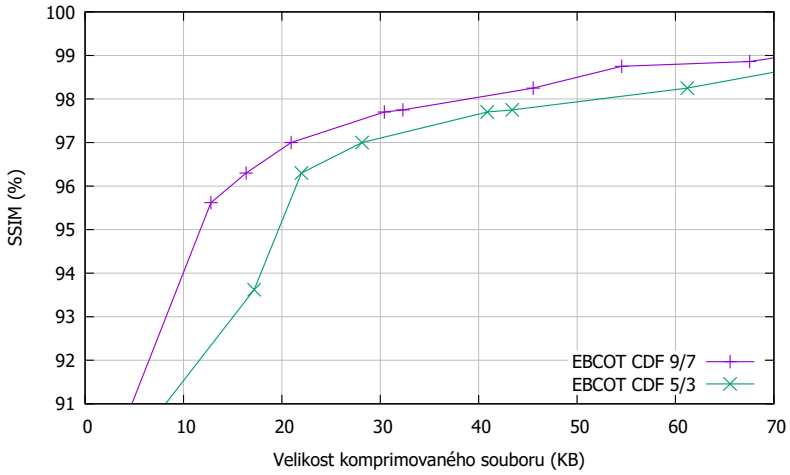
Graf 4.4 a 4.5 znázorňují výsledky srovnání pro trojici barevných testovaných obrázků. Jak lze z grafu vyčíst barevný prostor YCBCR má o dost lepší průběh než základní RGB. V průměru pro všechny tři testované vzorky má RGB o 30 procent větší bitový tok než YCBCR. Na základě těchto zjištění se v dalších testech s barevnými obrázky bude využívat výhradně prostor YCBCR.

Srovnání komprese dle volby vlnky

V dalším srovnání se zjistí účinnost vlnek CDF 9/7 a CDF 5/3. První vlnku standard JPEG2000 využívá pro ztrátovou kompresi, a druhou vlnku pro bezeztrátovou kompresi.



Obrázek 4.6: PSNR: Srovnání vlnek CDF 5/3 a CDF 9/7.

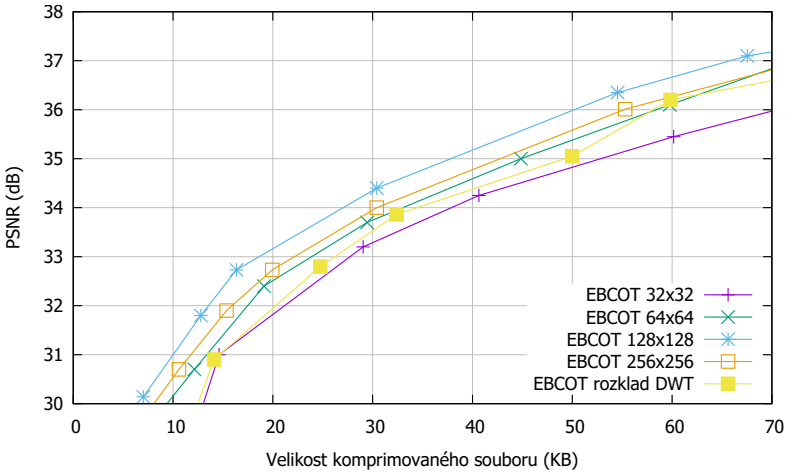


Obrázek 4.7: SSIM: Srovnání vlnek CDF 5/3 a CDF 9/7.

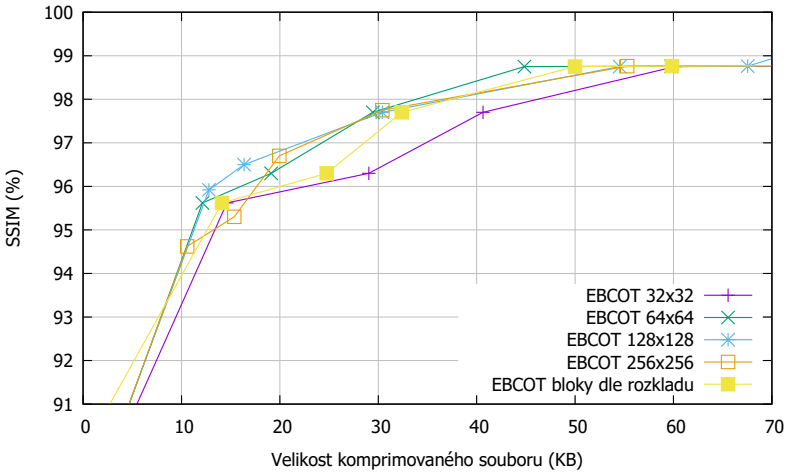
Jak se dalo předpokládat, tak grafy 4.6 a 4.7 potvrdily, že vlnka CDF 5/3 není tak účinná jako ztrátová vlnka CDF 9/7. V průměru při srovnání výsledků pro první tři vzorky je ztrátová vlnka o 15 procent účinnější, proto v dalším testování bude využita pouze vlnka CDF 9/7.

Srovnání komprese dle velikosti bloků EBCOT

U implementovaného algoritmu EBCOT lze zvolit velikost kódovacích bloků. Při srovnání byly zvoleny velikosti čtvercových bloků a to 32x32, 64x64, 128x128, 256x256. Poslední byly zvoleny velikosti bloků dle rozkladu DWT.



Obrázek 4.8: PSNR: Srovnání kvality komprese dle velikosti bloku u kódování algoritmem EBCOT.



Obrázek 4.9: SSIM: Srovnání kvality komprese dle velikosti bloku u kódování algoritmem EBCOT.

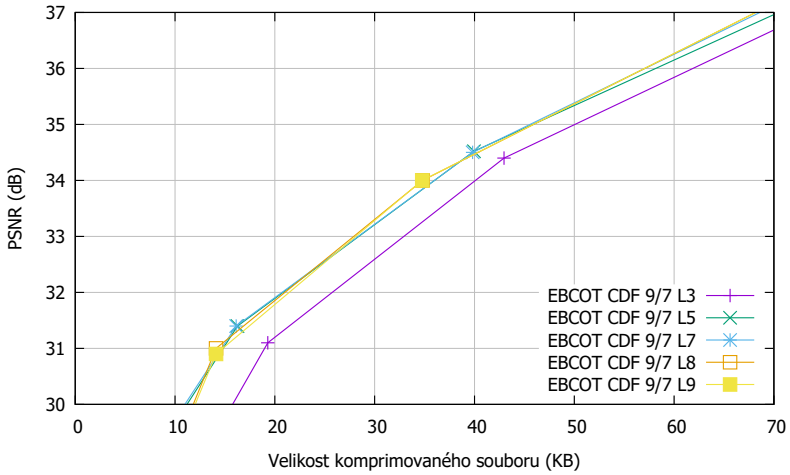
Velikost bloku 32x32 se ukázala jako nepříliš vhodná, mezi všemi testovanými skončila na posledním místě. Nejlépe dle srovnání dopadla velikost bloku 128x128, která měla v průměru mezi všemi testovanými obrázky nejlepší efektivitu. Ostatní velikosti bloků dosahují podobných průběhů. Proto se v dalším srovnáváním bude využívat velikost bloku 128x128, pokud nebude uvedeno jinak. Ve formátu JPEG2000 by se tato velikost bloku použít nedala, protože potřebuje škálovat tok paketů, u toho je potřeba co nejmenší velikost bloků.

Při vytváření knihovny a implementací možných rozšíření bylo vyzkoušeno při vlnkové transformaci rozkládat už transformované bloky typu LH, HL a HH z první úrovně rozkladu

do stejné úrovně jako blok LL. To nevedlo ke zlepšení efektivity komprese u větších bloků, ale u meších došlo k patrnému zlepšení. Hlavním přínosem bylo, že kvůli této upravě vzniklo více nulových rovin v blocích.

Srovnání komprese dle úrovně zanoření DWT rozkladu

Další otázkou je, jak ovlivňuje úroveň rozkladu DWT samotnou kompresi. Výsledný graf 4.10 zobrazuje průměrné hodnoty pro všechny testované vzorky. V grafu L3 označuje rozklad do nejmenší velikosti bloku 128x128 a například L7 zastupuje rozklad do velikosti bloku 8x8.



Obrázek 4.10: PSNR: Srovnání kvality komprese dle úrovně rozkladu DWT.

Výsledky grafu ukazují, že není u algoritmu EBCOT zapotřebí úplný rozklad vstupního obrazu. Při kompresi pro rozklad od úrovně L5 do L9 je rozdíl mezi jednotlivými výsledky v rámci desetin procenta.

Srovnání komprese dle typu algoritmu pro kódování koeficientů

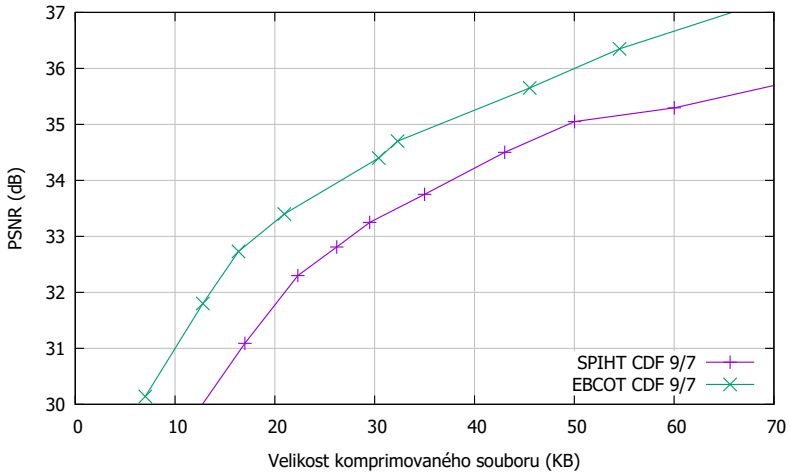
V následující části práce budou vyhodnocena závislost kvality komprese na zvoleném algoritmu pro kódování koeficientů vlnkové transformace. Budou vyhodnoceny pouze nejlepší implementace algoritmu SPIHT a EBCOT.

Z naměřených hodnot pro všechny vzorky z grafů 4.11 a 4.12 vyplývá , že nejlepší komprese dosahuje z implementovaných algoritmů v knihovně EBCOT. V průměru je lepší přibližně o 10 procent. Proto dále jako nejlepší algoritmus pro kódování koeficientů transformace je využíván pouze vítězný algoritmus EBCOT, který lépe zakódovává transformované koeficienty.

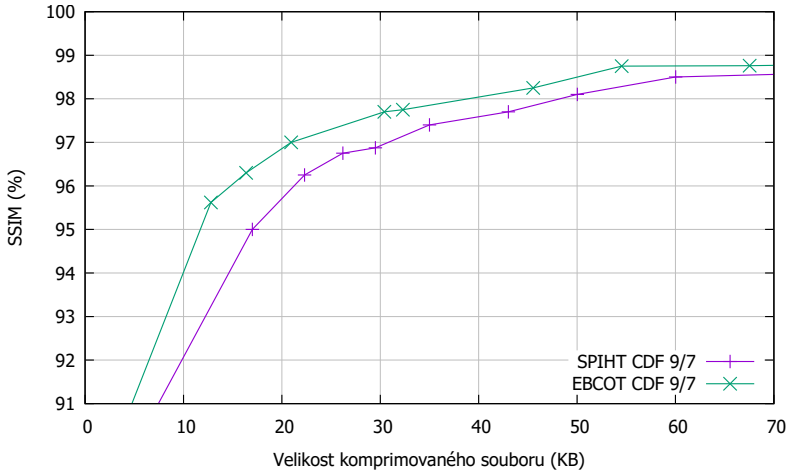
Srovnání komprese dle transformace

Nyní bude srovnána kvalita komprese vstupního obrazu dle zvolené transformace, nejdříve pro obrázek *Lena*.

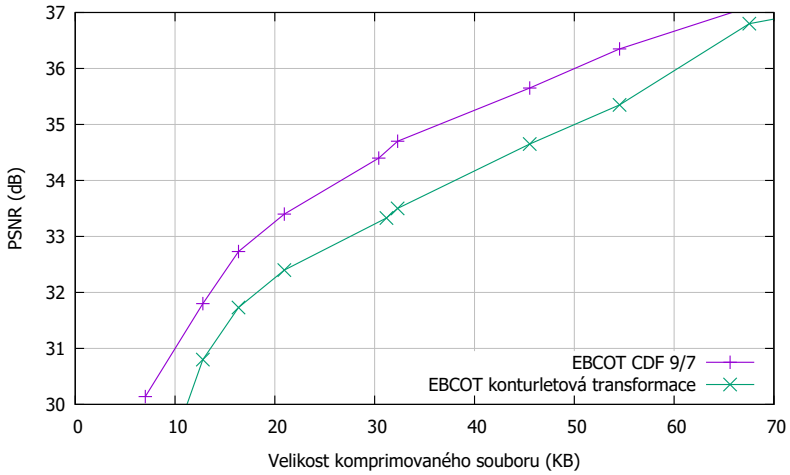
Pro tento reálný barevný obraz je jednoznačnou vhodnější diskrétní vlnková transformace dle zvolených metrik. Horší výsledek konturletové transformace může být zapříčiněn implementací transformace pro omezený počet směrů na jednotlivých úrovních rozkladu.



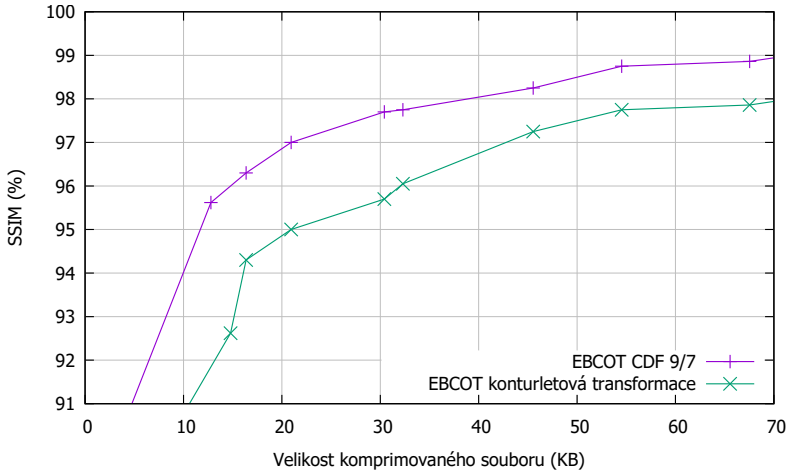
Obrázek 4.11: PSNR: Srovnání kvality komprese dle zvoleného algoritmu pro kódování koeficientů vlnkové transformace.



Obrázek 4.12: SSIM: Srovnání kvality komprese dle zvoleného algoritmu pro kódování koeficientů vlnkové transformace.

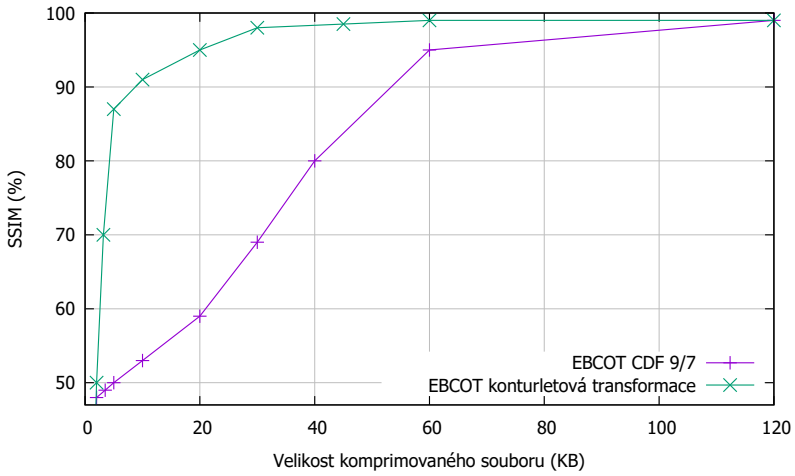


Obrázek 4.13: PSNR: Srovnání kvality komprese dle zvolené transformace pro obrázek *Lena*.



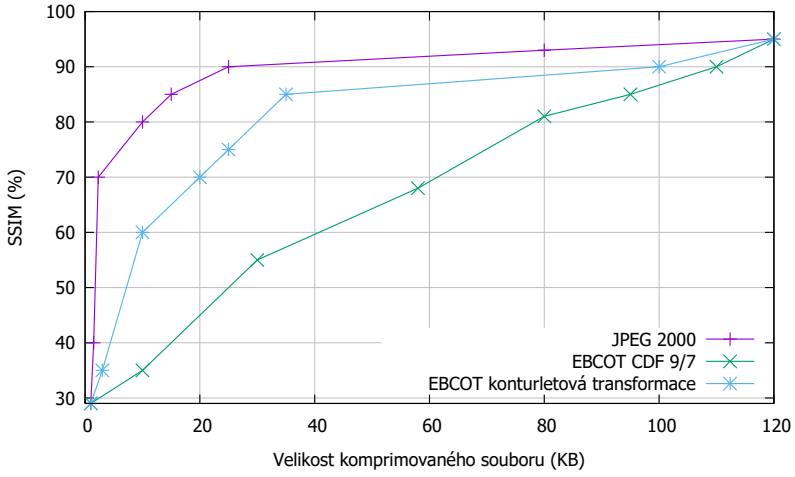
Obrázek 4.14: SSIM: Srovnání kvality komprese dle zvolené transformace pro obrázek *Lena*..

Nyní srovnáme transformace pro otisk prstů na obrázku4.1d. Využijeme k tomu pouze metriku SSIM.



Obrázek 4.15: SSIM: Srovnání kvality komprese dle zvolené transformace pro obrázek *Fingerprint1*.

Pro obrázek *Fingerprint1* má konturletová transformace lepší průběh než vlnková. Je to tím, že konturletová transformace lépe reprezentuje rysy ve vysokých frekvencích. Lze říci, že konturlové transformaci více vyhovují obrazy z více křivkami.

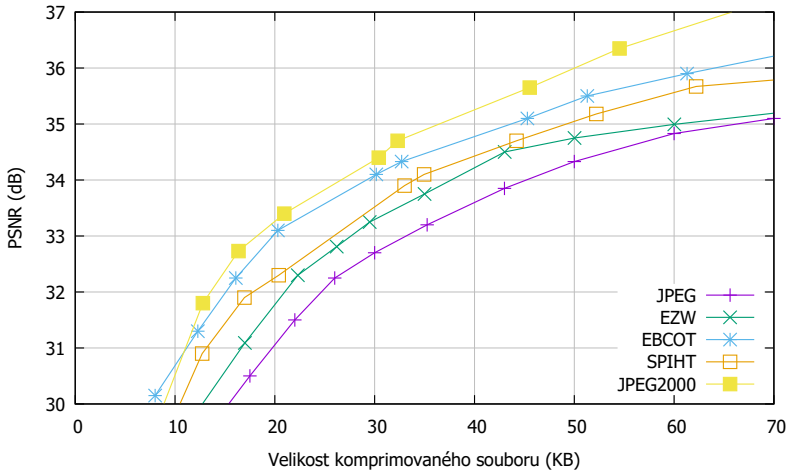


Obrázek 4.16: SSIM: Srovnání kvality komprese dle zvolené transformace pro obrázek *Fingerprint1* (umělý šum).

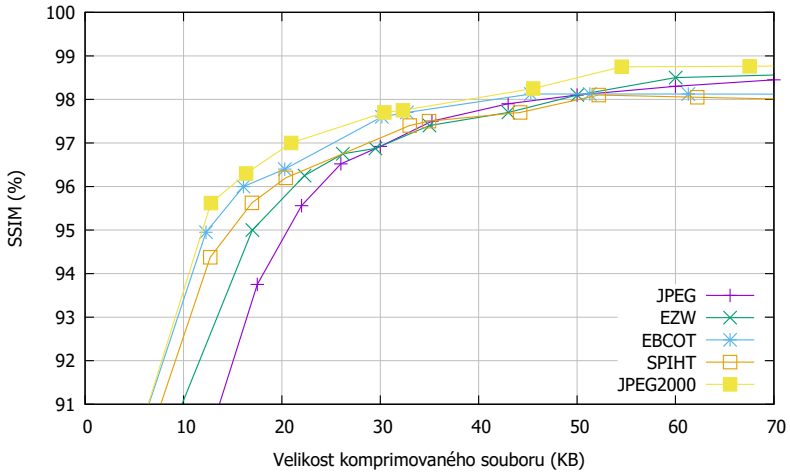
V grafu 4.16 dochází ke srovnání vlnkové transformace, konturletové transformace a formátu JPEG2000 pro uměle zašuměný obrázek *Fingerprint1*. Na první pohled je patrné, že nejlépe dopadl formát JPEG2000 následován konturletovou transformací. U vlnkové transformace dochází ke zhoršení kvality nejspíše opakováním významných hran v obraze. Řešením by bylo aplikovat v knihovně propracovanější nastavení výsledné kvality.

Celkové srovnání

V této podkapitole bude porovnána kvalita algoritmů implementovaných v knihovně současně se standardy JPEG a JPEG2000. U jednotlivých implementovaných algoritmů jsou pro vyhodnocení zvoleny jejich nejlepší možné parametry, pro co nejlepší a nejkvalitnější kompresi. Ani u jednoho z algoritmů nedochází k podvzorkování barevných složek, které by mohlo ve velké míře ovlivnit výsledky metriky SSIM.



Obrázek 4.17: PSNR: Celkové srovnání pro všechny vzorky.



Obrázek 4.18: SSIM: Celkové srovnání pro všechny vzorky

Grafy 4.17 a 4.18 zobrazují výsledky pro všechny vzorky. Mezi nejlepší patřil formát JPEG2000 (zastoupeným nástrojem KAKADU) a algoritmus EBCOT. Dále za nimi byly algoritmy SPIHT a EZW. Nejhůře dopadl nejvíce využívaný formát z testovaných a to JPEG.

4.3 Vizuální porovnání

Jak již bylo zmíněno v teoretické části metriky PSNR a SSIM nereflektují úplně přesně posuzování vzorků dle lidského vnímání. Proto je v práci uvedeno i subjektivní hodnocení výsledků pro určené metody z knihovny a formáty JPEG a JPEG2000.

Porovnání vizuální kvality výřezů obrázku *Lena*.

(a) JPEG (b) JPEG2000

(c) EBCOT konturletová transformace (d) EBCOT vlnka CDF 9/7

Obrázek 4.19: Výřezy obrázku *Lena* pro subjektivní porovnání.

Obrázky 4.19 byly získány vyříznutím z již komprimovaných obrazových dat pro obrázek *Lena*. Kde jednotlivé velikosti komprimovaných souborů jsou z okolí 7 kB.

U výřezu JPEG jsou jasně patrné blokové artefakty, které degradují výsledný vizuální vzhled komprimovaného vzorku. Výřez z konturletovou transformací nevykazuje známky blokových efektů, ale lze u hran spatřit jejich rozmazání. Při zaměření na formát JPEG2000 a EBCOT s využitím vlnky CDF 9/7 si lze povšimnout u prvního jmenovaného větší neostrost hran.

Porovnání vizuální kvality výřezů obrázku *Fruits*.

(a) JPEG (b) JPEG2000 (c) EBCOT vlnka CDF 9/7

Obrázek 4.20: Výřezy obrázku *Fruits* pro subjektivní porovnání.

U vzorku *Fruits* působí EBCOT a JPEG2000 v celku podobně oba nemají úplně ostré hrany, ale celkový dojem je velmi kvalitní. U standardního formátu JPEG lze opět spatřit blokové artefakty.

Porovnání vizuální kvality výřezů obrázku *Watch*.

(a) JPEG (b) JPEG2000 (c) EBCOT vlnka CDF 9/7

Obrázek 4.21: Výřezy obrázku *Watch* pro subjektivní porovnání.

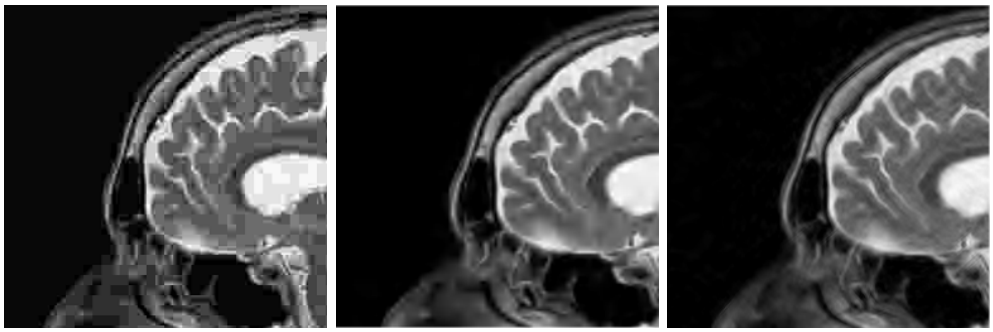
Pro obrázek s označením *Watch* jsou u formátu JPEG opět vidět blokové artefakty, ovšem co se týká ostrosti textu patří mezi nejlepší. Zatímco formát JPEG2000 s algoritmem EBCOT působí rozostřeným dojmem.

Porovnání vizuální kvality výřezů obrázku *Fingerprint1*.

Při porovnání výřezů obrázku 4.22 lze vidět, že JPEG má nejvíce ozubené hrany otisku. U JPEG2000 vypadají hrany papilárních linií rozostřeným dojmem. Zatímco algoritmus EBCOT s konturletovou transformací nabízí ostré hrany otisknutých linií prstu. Bohužel v okolí otisku přibývají nechtěné artefakty.

(a) JPEG (b) JPEG2000 (c) EBCOT kont. trans.

Obrázek 4.22: Výřezy obrázku *Fingerprint1* pro subjektivní porovnání.



(a) JPEG (b) JPEG2000 (c) EBCOT kont. trans.

Obrázek 4.23: Výřezy obrázku *Xray* pro subjektivní porovnání.

Porovnání vizuální kvality výřezů obrázku *Xray*.

U obrázků 4.23 lze opět vidět pro formát JPEG blokové artefakty, které ovšem nevypadají nijak moc rušivě. JPEG2000 a EBCOT jsou vizuálně srovnatelné, a lze u nich zaznamenat drobné rozmazání.

Porovnání vizuální kvality výřezů obrázku *Fingerprint2*.

(a) JPEG (b) JPEG2000 (c) EBCOT kont. trans.

Obrázek 4.24: Výřezy obrázku *Fingerprint2* pro subjektivní porovnání.

Na posledním testovacím vzorku, na kterém je fotografie otisku prstu dopadl nejlépe vizuálně algoritmus EBCOT s konturletovou transformací. U formátu JPEG2000 si lze všimnout rozmazání. U formátu JPEG je obrázek okupován blokovými artefakty.

Zhodnocení vizuálního porovnání

Při vizuálním porovnání byly zhodnoceny výřezy pro testované vzorky s velikosti komprimovaného souboru 7 kB. Ze vzniklých dat bylo vypozorováno, že nejvíce blokových artefaktů vzniká při využití formátu JPEG. Dalším zjištěním bylo, že algoritmus EBCOT a formát JPEG2000 mají tendenci při velké kompresním poměru u obrázkům s velkým počtem křivek jejich hrany rozostřit. Konturletová transformace, která byla implementovaná v knihovně, není vhodná pro reálná obrazová data, které potřebují různý směrový rozklad na jednotlivých rovinách dekompozice.

Kapitola 5

Závěr

Tato technická zpráva se skládá ze tři části. První kapitola se věnuje do detailů kompresí obrazu od vhodné volby barevného modelu, transformace, transformační vlnky až po jednotlivé algoritmy pro kódování transformovaných koeficientů. Navíc jsou v této části popsány techniky pro měření výsledné kvality komprese, které byly využity u implementace. Nejpodrobněji je zde popsána diskrétní vlnková transformace a algoritmus EBCOT, které byly stěžejní pro celou práci.

Druhá kapitola nazvaná Implementace pojednává o samotné realizaci jednotlivých modulárních bloků v knihovně. Nejprve je zde blokově popsán samotný návrh knihovny. Následuje zpráva o implementaci diskrétní vlnkové transformace s využitím techniky *lifting*. V dalším kroku je popsána implementace konturletové transformace. Tato kapitola dále obsahuje popis realizací algoritmů EZW, SPIHT a EBCOT, které se využívají pro kódování transformovaných koeficientů. V neposlední řadě se kapitola věnuje nastínění implementace MQ kodéru.

Třetí a poslední kapitola pojednává o srovnávání jednotlivých barevných prostorů, transformací, transformačních vlnek a kódovacích algoritmů. V dalším kroku jsou nejlepší možné modifikace implementovaných algoritmů srovnány ze standardy JPEG a JPEG2000. Závěrem této kapitoly je shrnutí vizuálního porovnání jednotlivých výsledků.

Zhodnocení výsledků

Ze srovnávání na testovaných vzorcích vyplynulo několik závěrů. Vhodnějším prostorem pro kompresi barevných obrazových dat je YCBCR. Vlnka CDF 9/7 je efektivnější při ztrátové kompresi než vlnka CDF 5/3. Pro reálný obraz, jako je na obrázku 4.1a, je vhodnější využít vlnkovou transformaci oproti konturletové, která zvýší velikost transformovaných dat až o 33 procent. Naopak pro obrázek 4.1f byla zjištěna jako vhodnější transformace v knihovně konturletová před vlnkovou, kde koeficienty těchto transformací byly zakódovány úplně stejným algoritmem. Nakonec byla konturletové transformace i pro tento typ obrazu překonána standardem JPEG2000 hlavně díky využití algoritmu EBCOT Tier2. V celkovém srovnání pro všechny testovací vzorky byl nejlepší standard JPEG2000, který byl nejvhodnější pro všechny jednotlivé obrázky, až na obrázek 4.1a, kde byl nejlepší algoritmus EBCOT. Ve vizuálním srovnání byl opět nejlepší formát JPEG2000.

Vlastní přínos

Jedním z přínosů bylo vytvoření kompresní knihovny, která umožňuje komprimovat pomocí různých transformací a různých algoritmů pro kódování koeficientů. Hlavním přínosem práce bylo porovnání a zhodnocení jednotlivých algoritmů pro kódování koeficientů vlnkové transformace. Navíc byla v knihovně implementována i konturletová transformace. Ta sloužila k dalšímu srovnávání kvalit vlnkové transformace. Knihovna je publikována na *github.com* k veřejnému využití pod názvem `compression_library`.

Za další přínos lze považovat samotná práce, která se snaží popisovat různé metody komprese obrazu a metody pro porovnání kvality komprese. V neposlední řadě slouží ke srovnání jednotlivých komprimačních řetězců.

Další vývoj

V oblasti zabývající se kompresí obrazu se neustále hledají nové způsoby jak komprimovat obrazová data, tak aby zabíral co nejméně místa, a zároveň byl co nejvíce kvalitní. Dalším směrem jakým by se další práce mohla zabývat by mohlo být využít implementované transformace, například k odstranění nedostatků obrazu jako je rozmazání, zašumění a jiné.

Pokud by se mělo dále pokračovat pouze v oblasti komprese, dalším možným krokem by byla implementace sofistikovanějšího kodéru pro zakódování bitového toku nebo rozšíření algoritmů pro zakódování transformovaných koeficientů. V neposlední řadě by šlo srovnat u komprese například odolnost kódovaní proti chybám v přenosovém řetězci, bezpečnost při přenosu, možnost náhodného přístupu do zakódovaného souboru dat a objektový způsob práce.

Další možností by bylo otestovat jiné transformace nebo nějakým způsobem propojit transformace do jedné, která by využívala hlavně výhod jednotlivých z nich.

V neposlední řadě by šlo algoritmy využít pro kompresi snímků ve videu, nejdřív by se musel nejspíš výpočet implementovat na grafické kartě pro rychlejší běh.

Literatura

[1] Cohen, A.; Daubechies, I.; Feauveau, J.-C.: Biorthogonal bases of compactly supported wavelets. *Communications on Pure and Applied Mathematics*, ročník 45, č. 5, 1992: s. 485–560, ISSN 1097-0312, doi:10.1002/cpa.3160450502.
URL http://dx.doi.org/10.1002/cpa.3160450502

[2] Delaunay, X.: EBCOT coding passes explained on a detailed example. 2010.
URL http://d.xav.free.fr/ebcot/EBCOT_example.pdf

[3] Delaunay, X.; Chabert, M.; Morin, G.; aj.: Bit-Plane Analysis and Contexts Combining of JPEG2000 Contexts for On-Board Satellite Image Compression. In *2007 IEEE International Conference on Acoustics, Speech and Signal Processing - ICASSP '07*, ročník 1, April 2007, ISSN 1520-6149, s. I–1057–I–1060, doi:10.1109/ICASSP.2007.366093.

[4] Do, M. N.; Vetterli, M.: Contourlets: a directional multiresolution image representation. In *Proceedings. International Conference on Image Processing*, ročník 1, 2002, ISSN 1522-4880, s. I–357–I–360 vol.1, doi:10.1109/ICIP.2002.1038034.

[5] Do, M. N.; Vetterli, M.: The contourlet transform: an efficient directional multiresolution image representation. *IEEE Transactions on Image Processing*, ročník 14, č. 12, Dec 2005: s. 2091–2106, ISSN 1057-7149, doi:10.1109/TIP.2005.859376.

[6] Kaše, D.: Komprese obrazu pomocí vlnkové transformace. 2015.

[7] Klejmová, E.: Měření kvality pro HEVC. 2014.

[8] Mallat, S.; with contributions from Gabriel Peyré.: *A wavelet tour of signal processing the sparse way*. Amsterdam: Elsevier/Academic Press, třetí vydání, 2009, ISBN 9780080922027.

[9] Owen, T.; Hauck, S.; Owen, T.; aj.: Arithmetic Compression on SPIHT Encoded Images. Department of Electrical Engineering University of Washington, 2002.
URL https://www2.ee.washington.edu/techsite/papers/documents/UWEETR-2002-0007.pdf

[10] Said, A.; Pearlman, W. A.: *A New Fast/Efficient Image Codec Based on Set Partitioning in Hierarchical Trees*. Boston, MA: Springer US, 2002, ISBN 978-0-306-47043-1, s. 157–170, doi:10.1007/0-306-47043-8_9.

[11] Shapiro, J.: Embedded Image Coding Using Zerotrees of Wavelet Coefficients. *Trans. Sig. Proc.*, ročník 41, č. 12, Prosinec 1993: s. 3445–3462, ISSN 1053-587X,

doi:10.1109/78.258085.
URL http://dx.doi.org/10.1109/78.258085

[12] Taubman, D.: Block entropy coding in embedded block coding with optimized truncation image compression. Září 2 2008, uS Patent 7,421,137.
URL http://google.si/patents/US7421137

[13] Urbánek, P.: Komprese obrazu pomocí vlnkové transformace. 2013.

[14] Vetterli, M.: *Multirate Filter Banks for Subband Coding*. Boston, MA: Springer US, 1991, ISBN 978-1-4757-2119-5, s. 43–100, doi:10.1007/978-1-4757-2119-5_2.
URL http://dx.doi.org/10.1007/978-1-4757-2119-5_2

[15] Wallace, G. K.: The JPEG Still Picture Compression Standard. *Commun. ACM*, ročník 34, č. 4, Duben 1991: s. 30–44, ISSN 0001-0782, doi:10.1145/103085.103089.
URL http://doi.acm.org/10.1145/103085.103089

[16] Wang, M.; Xiong, Y.: Embedded quadtree wavelets in image compression. Červenec 12 2005, uS Patent 6,917,711.
URL https://www.google.ch/patents/US6917711

[17] Wang, Z.; Bovik, A. C.; Sheikh, H. R.; aj.: Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, ročník 13, č. 4, April 2004: s. 600–612, ISSN 1057-7149, doi:10.1109/TIP.2003.819861.

Přílohy

Příloha A

Obsah přiloženého paměťového média

CD nosič obsahuje:

- Diplomovou práci ve formátu PDF.
- Zdrojové soubory pro opětovné vytvoření diplomové práce ve složce `dip_src`.
- Zdrojové soubory pro překlad knihovnu ve složce `src`.
- Video vytvořené k diplomové práci.