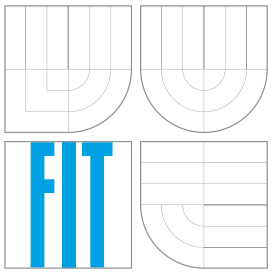
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOMPRESE OBRAZU POMOCÍ VLNKOVÉ TRANSFORMACE

IMAGE COMPRESSION USING THE WAVELET TRANSFORM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. DAVID KAŠE

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. DAVID BAŘINA

BRNO 2015

Abstrakt

Tato práce se zabývá kompresí obrazů pomocí vlnkové, konturletové a shearletové transformace. V první části je nastíněna problematika komprese obrazu a způsobu měření kvality. Jsou zde popsány pojmy vlnka, multirozklad a měřítková funkce, detailněji je dále představeny jednotlivé transformace. Z algoritmů pro kódování koeficientů jsou uvedeny EZW, SPIHT a okrajově EBCOT. V druhé části je popsán způsob návrhu a implementace knihovny. V poslední části jsou srovnány jednotlivé transformace mezi sebou a s formátem JPEG 2000. Výsledkem práce byl určen typ obrazu, u kterého dosahovala implementovaná konturletová a shearletová transformace lepších výsledků než vlnková. Formát JPEG 2000 nebyl překonán.

Abstract

This thesis deals with image compression using wavelet, contourlet and shearlet transformation. It starts with quick look at image compression problem a quality measurement. Next are presented basic concepts of wavelets, multiresolution analysis and scaling function and detailed look at each transform. Representatives of algorithms for coeficients coding are EZW, SPIHT and marginally EBCOT. In second part is described design and implementation of constructed library. Last part compare result of transforms with format JPEG 2000. Comparison resulted in determining type of image in which implemented contourlet and shearlet transform were more effective than wavelet. Format JPEG 2000 was not exceeded.

Klíčová slova

Ztrátová komprese, diskrétní vlnková transformace, konturletová transformace, shearletová transformace, EZW, SPIHT, JPEG 2000

Keywords

Lossy compression, discrete wavelet transform, contourlet transform, shearlet transform, EZW, SPIHT, JPEG 2000

Citace

David Kaše: Komprese obrazu pomocí vlnkové transformace, diplomová práce, Brno, FIT VUT v Brně, 2015

Komprese obrazu pomocí vlnkové transformace

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny

.....
David Kaše
26. května 2015

Poděkování

Chtěl bych poděkovat svému vedoucímu, jímž byl Ing. David Bařina, za jeho rady s postupem práce, když jsem si sám nebyl jist.

Obsah

Kapitola 1

Úvod

Komprese obrazu je velice důležitou součástí skoro každého formátu, se kterým je možné se setkat. Uchovávat obraz rozumné velikosti bez jakékoliv komprese nemá v dnešní době smysl. Pro příklad stacionární obraz ve full HD rozlišení, tedy 1920×1080 pixelů, obsahuje dohromady zhruba dva miliony pixelů, dále je každý pixel kódován například barevným prostorem RGB24, což odpovídá třem bytům na jeden pixel. Celkově tedy jeden full HD obraz bez jakékoliv komprese zabere v paměti zhruba 6 MB. Komprese obrazu se nepromítá pouze na stacionární obrazy, ale velký vliv má logicky i na sekvenci obrazů, neboli video, kde běžně připadne třicet snímků za sekundu. Z uvedeného příkladu je vidět, že komprese obrazu je velmi významnou oblastí.

Existují různé přístupy ke kompresi obrazu. Tato práce se zabývá přístupy vycházející z pohledu na obraz jako na dvourozměrný signál. Mezi nejznámější zástupce této kategorie lze zařadit populární obrazový formát JPEG dle specifikace [13], který využívá dvourozměrnou variantu kosinové transformace. Další iterací, která vzešla z formátu JPEG, je JPEG 2000 využívající vlnkovou transformaci, u níž bylo dokázáno, že produkuje lepší výsledky než dříve zmíněný JPEG v článku [11]. Jako alternativní možnost je zde uvedena komprese stacionárního obrazu konturletovou a shearletovou transformací.

V následující druhé kapitole je definován pojem obraz a z čeho je složen. Je zde také nastíněna problematika barevných prostorů společně s popisem RGB a $\mathrm{YC}_B\mathrm{C}_R$. Dále jsou definovány základní principy komprese obrazu a základní metody měření kvality komprese.

Následující třetí kapitola se zabývá diskrétními transformacemi, konkrétně jsou definované základní pojmy jako vlnky, měřítková funkce a multirozklad. Dále v kapitole jsou uvedeny tři diskrétní transformace. Jedná se o diskrétní vlnkovou transformaci, konkrétně její dvourozměrnou diskrétní variantu. Druhou transformací je zde uvedena konturletová transformace. Popis konturletové transformace je zde uveden pomocí multirozkladu a směrových filtrů. Pro multirozklad je použita Laplaceova pyramida a kombinací multirozkladu a směrových filtrů je dosažena konturletová banka filtrů. Poslední uvedenou transformací je shearletová, která pracuje s podobným principem jako předchozí konturletová. Poslední část této kapitoly je věnována algoritmům pro kompresi koeficientů. Jsou zde popsány algoritmy pro kódování koeficientů EZW, SPIHT a okrajově také EBCOT.

Další kapitola je věnována popisu návrhu a implementace knihovny spolu s demonstrační aplikací. Implementační část je rozdělena do logicky souvisejících bloků a podrobně popsána komunikace uvnitř těchto bloků.

Předposlední kapitola obsahuje postup srovnávání naměřených hodnot určení efektivity různých výpočetních kombinací na daných typech obrazu. Dále jsou naměřené výsledky srovnány s existujícím formátem JPEG 2000. Porovnány jsou také vlastnosti transformací

v podobě tvořených artefaktů při vyšší kompresi obrazu.

Poslední kapitola obsahuje závěr této práce se zhodnocením dosažených výsledků a návrhu dalšího možného postupu na základě této práce.

Praktická část této práce obsahovala implementaci knihovny poskytující kompresi pomocí třech uvedených transformací. Na výsledné implementaci bylo provedeno srovnání transformací s jednotlivými algoritmy pro kódování koeficientů na různých typech obrazu. Bylo dosaženo efektivnější komprese (vzhledem k vlnkové transformaci) pomocí konturletové a shearletové transformace pouze na určitém typu obrazu.

Kapitola 2

Komprese obrazu

Komprese obrazu je postup pro odstranění nebo zmenšení množství redundantních informací z reprezentace obrazu. Hlavním důvodem často bývá efektivnější využití paměti za účelem uložení nebo přenosu. V této kapitole si přiblížíme, co to je digitální obraz, princip barevného modelu a princip komprese obrazů společně se způsoby měření kvality a efektivity vlastní komprese.

Digitální obrazy rozdělujeme na dvě skupiny podle způsobu reprezentace obsahu. Vektorové obrazy nesou informace o matematicky definovaných tělesech, jako jsou úsečky, elipsy, křivky atd. Výsledný obraz je postupně skládán z těchto objektů. Vektorové obrazy obsahují malou míru redundance a jejich komprese případně spočívá v kompresi samotném zápisu reprezentace objektů.

Mnohem častější jsou rastrové obrazy. Tyto obrazy jsou reprezentovány (2D, případně 3D) maticí obsahující informace o barvě příslušné části obrazu (pixelu, případně voxelu ve 3D). Tato reprezentace obrazu je oproti předcházející paměťově mnohem náročnější a proto je často provázena nějakou kompresní metodou. Tato práce se věnuje kompresi rastrových obrazů.

2.1 Barevné modely

Hodnoty jednotlivých pixelů v obraze (voxelů v prostoru) se vztahují k určitému barevnému modelu. Jednotlivé barevné modely si lze představit jako souřadnicový prostor (typicky 2D nebo 3D), kde jednotlivé barvy jsou body v tomto prostoru. Pouhá volba vhodného barevného modelu již může vést ke kompresi, neboť lze podvzorkovat části barevného modelu, jenž mají na výslednou kvalitu minimální vliv. Barevný model RGB, založený na míchání barev červené, zelené a modré, je typický pro vnímání a reprezentaci obrazů v elektronických zařízení. Pro digitální reprezentaci obrazu je často použit barevný model $\mathrm{YC}_B\mathrm{C}_R$, jenž zohledňuje vlastnosti lidského zraku, jako je vyšší citlivost na jasovou složku barvy než samotnou barvu. Mezi další barevné modely patří například HSV (**h**ue, **s**aturation, **v**alue) a HSL (**h**ue, **s**aturation, **l**ightness), jenž jsou založeny na sytosti a odstínu barev.

RGB

Barevný model RGB je založen na aditivním míchání barev. Každá barva je tedy složena ze tří základních barev světla, jedná se o barvy červené, zelené a modré. Barevný model je možné si představit ve třírozměrném prostoru, kde každá ze tří barev je reprezentována jednou osou. V počítačové technice se velmi často setkáme s barevným modelem RGB24,

jedná se o RGB model s vyznačenou celkovou velikostí všech tří složek, přičemž typicky na každou složku přísluší 8bitů. U RGB24 má každá barva 256 hodnot a mícháním těchto tří barev dostáváme 256^3, tj. 16,7 miliónů barev. Existují i jiné varianty RGB (například červená a zelená barva mají o jeden bit větší rozsah než barva modrá), ale žádná jiná varianta není tak rozšířená a používají se pouze ve specializovaných přístrojích. S rostoucím vývojem vznikla potřeba pro průhlednost barev a z barevného modelu RGB vznikl RGBA, který má přidanou čtvrtou novou složku, jenž určuje průhlednost dané barvy. RGBA není barevným modelem v pravém slova smyslu, jde pouze o uložení barvy RGB s přidanou informací o průhlednosti (alfa kanálu).

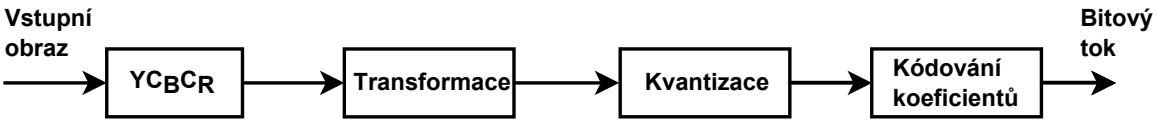
$\mathbf{YC_BC_R}$

Jedná se o diskrétní barevný model zohledňující vlastnosti lidského zraku. Analogová verze se nazývá YP_BP_R, nicméně se i pro analogovou verzi občas používá označení YC_BC_R, případně $Y'C_BC_R$. Umožňuje oddělit jasovou složku od barev specifikované modrou a červenou chrominační složkou. Převody podle specifikace standardu JFIF (JPEG File Interchange Format) [13] mezi barevnými modely RGB24 a YC_BC_R a naopak jsou definovány v rovnicí (2.1).

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

2.2 Principy komprese

Komprese obrazu je typicky snahou o efektivnější využití paměti uchováním pouze důležitých, neredundantních informací o obraze. Ke kompresi jsou často využívány poznatky z biologie o lidském zraku, neboť tyto poznatky nám umožňují jednoduše aplikovat různé úpravy pro typicky zhoršení kvality obrazu beztoho, aby lidské oko poznalo rozdíl. Jako další užitečný faktor lze zařadit, co obrazy obsahují. Typicky se jedná o objekty reálného světa (nebo jim velice podobné), u kterých lze nalézt při zobrazení určité korelace a využít je ke kompresi. Existuje několik přístupů ke kompresi, ale v této práci se budeme zabývat transformacemi, konkrétně vlnkovými.



Obrázek 2.1: Blokové principiální schéma řetězce pro kompresi obrazu.

Postup komprese s využitím transformací se obvykle skládá z několika kroků (viz obrázek 2.1). Před aplikací transformace se obvykle obraz převádí do jiného vhodnějšího barevného modelu. Následně se obraz rozdělí na segmenty (může být i celý obraz jako jeden segment) a aplikuje se na ně daná transformace. Například u formátu JPEG je použita diskrétní kosinová transformace a u formátu JPEG 2000 je použita vlnková transformace. Transformací se typicky zamění informace v segmentech za koeficienty získané z transformace. Tyto koeficienty obsahují mnohem menší míru korelace než původní obrazová data. Na koeficienty se dále mohou aplikovat kvantizační tabulky. Posledním krokem je už pouze kódování do výstupního bitového toku.

Aby bylo možné takto komprimovaný obraz znovu zobrazit, je potřeba použít dekompresor. Ten je typicky zrcadlovým obrazem kompresoru a obsahuje inverzní funkce komprese. Postup dekodéru je tedy rozkódování bitového toku, aplikace inverzní transformace s následným poskládání jednotlivých bloků a převedením do barevného modelu vhodným pro zobrazení (například RGB).

Ztrátová a bezeztrátová komprese

Rozlišujeme mezi dvěma základními typy komprese na základě přístupu k rekonstrukci. Jako první je bezeztrátová komprese, u které jsou odstraněny pouze redundantní informace a zpětnou dekompresí lze sestavit původní obraz. V dříve popsaném postupu je typicky vynechána kvantizace, neboť je vždy ztrátová a nevratná. Druhým typem je ztrátová komprese. U této komprese jsou odstraněny redundantní informace společně s určitou částí nejméně významnějších neredundantních informací. Ztrátová komprese často zakládá na lidském vnímání obrazu, například poznatku, že lidské oko je více citlivé na změnu jasu než barvy.

2.3 Způsoby měření kvality

Pro vyhodnocení kvality komprese nestačí pouze kompresní poměr nebo faktor. Důležitým měřítkem je také kvalita komprimovaného obrazu a původního, respektive jejich rozdíly v kvalitě. U bezeztrátových kompresí nemá moc smysl měřit kvalitu komprimovaného obrazu, neboť je vždy stejná jako v původním obraze (hlavním principem bezeztrátové komprese je, že lze znovu sestavit původní obraz). U ztrátové komprese je naopak vhodné sledovat, nakolik se změnila kvalita obrazu, aby bylo možné určit jaká ztrátovost je příliš velká a v obraze se ztrácí klíčové aspekty.

Metody na určení kvality rozlišujeme na subjektivní a objektivní. Subjektivní metody spoléhají na člověka pro vyhodnocení kvality původního a pozměněného obrazu. Tyto metody mají výhodu v tom, že zpětná vazba je nejvěrohodnější pro ostatní (vyhodnoceno lidským zrakem). Nevýhodou je ale pomalé, drahé a nekonzistentní vyhodnocování. Objektivní metody jsou obvykle založeny na matematickém výpočtu. Výhody a nevýhody jsou pravým opakem subjektivních metod. Vyhodnocení je rychlé, levné a konzistentní, nicméně nevýhodou je špatná aproximace s vlastnostmi lidského zraku.

Špičkový poměr signálu k šumu

Špičkový poměr signálu k šumu, angl. Peak signal to noise ratio (zkr. PSNR), patří mezi nejzákladnější objektivní metody pro měření kvality signálu. Jedná se o vyjádření poměru mezi maximální možnou hodnotou signálu a energií rušivého signálu. Z důvodů dynamické charakteristiky signálů je PSNR často vyjádřen v logaritmickém měřítku (jednotky jsou poté decibely). Matematický vztah pro výpočet PSNR je zapsán rovnicemi (2.2).

$$\mathrm{PSNR} = 20 \log_{10} \left(\frac{\mathrm{MAX}_f}{\sqrt{\mathrm{MSE}}} \right) \tag{2.2}$$

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

Jak je vidět v (2.2) je k výpočtu PSNR potřeba zjistit střední kvadratickou odchylku (Mean square error - MSE). Lze ji vyčíslit podle (2.3), kde f je matice původního obrazu

a g je matice pozměněného/komprimovaného obrazu, m, resp. n, představuje počet řádků, resp. sloupců, v matici obrazu a i, resp. j, je řádkovým, resp. sloupcovým, indexem. MAX_f je maximální hodnota v originálním obraze (matici). Jelikož je střední kvadratická odchylka ve jmenovateli, tak PSNR není použitelná na shodné obrazy (MSE je rovno nule => dělení nulou v (2.2)).

Strukturální podobnost

Strukturální podobnost v článku [36], z angl. structural similarity (dále jen SSIM), je metoda srovnávající dva obrazy, která dobře koreluje s lidským vnímáním. Samotná metoda se skládá ze třech výpočtů. Jednotlivé výpočty reprezentují podobnosti jasu, kontrastu a struktury.

Prvním výpočtem je zjištění podobnosti jasu dvou obrazů. Střední hodnota jasu v obrazu je získána váženým průměrem přes celý obraz, vyjádřeno rovnicí (2.5).Matematický vztah pro výpočet jasové podobnosti $l(\mathbf{x}, \mathbf{y})$ je dán rovnicí (2.4). V rovnici (2.6) je L dynamickým rozptylem (pro 8 bitů je 255) a K_1 je malá konstanta ($K_1 \ll 1$).

$$l(\mathbf{x}, \mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.4}$$

$$\mu_x = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{2.5}$$

$$C_i = (K_i L)^2 \tag{2.6}$$

Druhý výpočet SSIM pro kontrast je započat úpravou vstupních obrazů odstraněním průměrného jasu, neboli upravený obraz je dán jako $\mathbf{x} - \mu_x$. Jako odhad kontrastu obrazu je použita směrodatná odchylka, rovnice (2.7). Porovnání kontrastu $c(\mathbf{x}, \mathbf{y})$ je porovnáním σ_x a σ_y, rovnice (2.8). Konstanta C_2 je opět dána podle rovnice (2.6).

$$\sigma_x = \left(\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)^2\right)^{\frac{1}{2}} \tag{2.7}$$

$$c(\mathbf{x}, \mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.8}$$

Posledním výpočtem je strukturální podobnost $s(\mathbf{x}, \mathbf{y})$. Obraz je dále podělen (normalizován) svoji směrodatnou odchylkou. Porovnávají se tedy obrazy $(\mathbf{x} - \mu_x)/\sigma_x$ a $(\mathbf{y} - \mu_y)\sigma_y$, rovnice (2.9). Stejně jako u předchozích dvou výpočtů je konstanta C_3 dána rovnicí (2.6). Člen σ_{xy} je definován rovnicí (2.10).

$$s(\mathbf{x}, \mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.9}$$

$$\sigma_{xy} = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y) \tag{2.10}$$

Kombinace všech třech výpočtů dává metodu SSIM v obecné rovnici (2.11). Pomocí zjednodušení $C_3 = C_2/2$ dostaneme rovnici (2.12). Konstanty C_1 a C_2 zajišťují stabilitu

metody pro případy, kdy se $(\mu_x^2 + \mu_y^2)$ nebo $(\sigma_x^2 + \sigma_y^2)$ blíží k nule.

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = f(l(\mathbf{x}, \mathbf{y}), c(\mathbf{x}, \mathbf{y}), s(\mathbf{x}, \mathbf{y})) \tag{2.11}$$

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2.12}$$

U barevných obrazů se často uvažuje pouze luminační složka. Výsledná matice udává na každém indexu koeficient podobnosti. Pro snadnější porovnání dvou obrazů se vypočítá průměr celé matice. Takto získaná výsledná hodnota se pohybuje v intervalu od -1 do 1, kde hodnota 1 je dosažena pouze pro shodné obrazy.

Kapitola 3

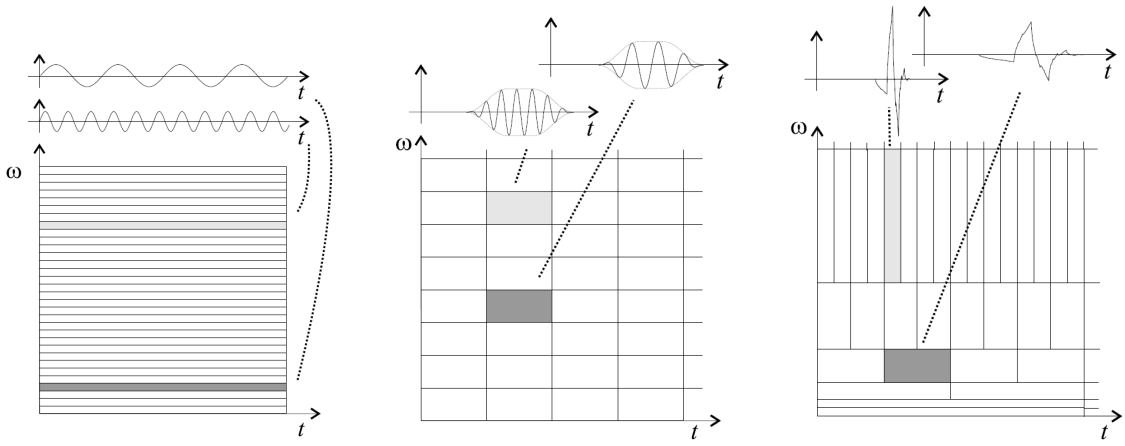
Diskrétní transformace

Tato kapitola se věnuje teoretickému rozboru třech diskrétních trasnformací. Prvně jsou uvedeny důležité pojmy vlnky, měřítkové funkce a multirozkladu, které jsou využity při dalším popisu. Vybrané transformace jsou diskrétní vlnková, konturletová a shearletová. Konec kapitoly je věnován algoritmům pro kódování koeficientů. Uvedeny jsou algoritmy EZW a SPIHT a okrajově zmíněn také algoritmus EBCOT.

Transformace signálu, ať už Fourierova, kosinová nebo vlnková, rozloží vstupní signál na popis pomocí jiného signálu (například komplexní exponenciála nebo kosinusovka). To nám umožňuje se dívat na původní signál z jiného pohledu. Fourierova transformace poskytuje dobrý pohled na zastoupené frekvence v signálu, jak je na obrázku 3.1(a). V tomto grafu ovšem chybí informace o umístění těchto frekvencí v čase. Existuje varianta Fourierovy transformace zvaná krátkodobá Fourierova transformace (obrázek 3.1(b)), jenž udává zastoupení frekvencí v časových oknech konstantní velikosti. Vlnková transformace je ještě o krok dále tím, že časová i frekvenční okna nemají konstantní velikost a jsou přizpůsobeny vstupnímu signálu, viz obrázek 3.1(c). Delší časová okna umožňují přesnější určení frekvencí, ale zhoršuje se tím přesnost časové lokality. Kratší časová okna poté mají lepší časovou přesnost, ale horší frekvenční rozlišení. Tento jev se také nazývá Heisenbergův princip neurčitosti [21], v tomto případě jsou dvěma veličinami čas a frekvence. U vlnkových transformací se tohoto principu využívá pro lepší frekvenční rozlišení u nižších frekvencí a vyšší přesnosti časové lokalizace u vyšších frekvencí, jak je znázorněno na obrázku 3.1(c).

Vlnková transformace je obecně definována rovnicí (3.1) na spojitém čase se dvěma reálnými parametry. Parametr s představuje posun a parametr r představuje dilataci. Dále $\psi^*_{r,s}(t)$ je komplexně sdružená vlnka.

Podobně jako u Fourierově transformaci se vstupní signál $f(t)$ převede do množiny bázových funkcí. Komplexní exponenciály u Fourierovy transformace a vlnky (wavelets) u vlnkové transformace. Způsob tvorby těchto vlnek je popsán rovnicí (3.2), kde $\frac{1}{\sqrt{r}}$ slouží k normalizaci energie. Vlnková transformace nedefinuje přesnou funkci pro vlnku, nýbrž umožňuje dosadit vlnku vhodnou pro daný signál. Inverzní podoba transformace je vyjádřena rovnicí (3.3). Diskretizace spojité vlnkové funkce spočívá v diskretizaci parametrů posunu s a dilatace r.



(a) Fourierova transformace (b) Krátkodobá Fourierova transformace (c) Vlnková transformace

Obrázek 3.1: Pokrytí frekvenční domény.

$$\mathrm{W}f(r,s) = \int f(t)\psi^*_{r,s}(t)\mathrm{d}t \tag{3.1}$$

$$\psi_{r,s}(t) = \frac{1}{\sqrt{r}}\psi\left(\frac{t-s}{r}\right) \tag{3.2}$$

$$f(t) = \frac{1}{C_\psi}\int_0^{+\infty}\int_{-\infty}^{+\infty}\mathrm{W}f(r,s)\psi_{r,s}(t)\mathrm{d}s\frac{\mathrm{d}r}{s^2} \tag{3.3}$$

3.0.1 Vlnky

Vlnková transformace původní signál rozkládá pomocí vlnek (angl. wavelets). Všechny vlnky použité v transformaci jsou odvozeny od jedné mateřské vlnky. Tyto dceřiné vlnky (3.5) jsou vyrobeny z mateřské vlnky pomocí posunu a dilatace, jak je uvedeno v rovnici (3.2).

Aby vlnka mohla být mateřskou, musí splňovat určité podmínky [21] [3]. První podmínkou je nulový průměr a jednotkovou plochu, rovnice (3.4). Jako poslední podmínka je tzv. přípustnost podle rovnice (3.6). Z podmínky přípustnosti a nulového průměru poté vyplývá, že pro nulovou frekvenci platí (3.7), kde $\widehat{\psi}(\omega)$ představuje Fourierovu transformaci vlnky ψ.

$$\int_{-\infty}^{+\infty}\psi(t)\mathrm{d}t = 0, \quad \int_{-\infty}^{+\infty}|\psi(t)|^2\mathrm{d}t = 1 \tag{3.4}$$

$$D = \left\{\psi_{r,s}(t) = \frac{1}{\sqrt{r}}\psi\left(\frac{t-s}{r}\right)\right\}_{s\in\mathbb{R},r\in\mathbb{R}^+} \tag{3.5}$$

$$C_\psi = \int_0^{+\infty}\frac{|\widehat{\psi}(\omega)|^2}{\omega}\mathrm{d}\omega < +\infty \tag{3.6}$$

$$|\widehat{\psi}(\omega)|^2_{\omega=0} = 0 \tag{3.7}$$

Vlnky, které splňují podmínky popsané v předcházející části, mohou disponovat dalšími vlastnostmi přímo ovlivňujícími proces komprese, ať už efektivitou nebo výpočetní náročností. Mezi tyto vlastnosti patří například existence nosiče, nulové momenty, regularita, neboli hladkost, a symetrie.

3.0.2 Měřítková funkce

Obecně počet posunutí a dilatací vlnek je při transformaci nekonečný. Jelikož nemá moc smysl počítat nekonečný počet dilatací, tak můžeme od určitého bodu popsat zbylé dilatace pomocí tzv. měřítkové (škálovací) funkce [21] označována ϕ v rovnici (3.8).

$$|\widehat{\phi}(\omega)|^2 = \int_1^{+\infty} |\widehat{\psi}(r\omega)|^2 \frac{\mathrm{d}r}{r} \tag{3.8}$$

3.0.3 Multirozklad

Doposud jsme uvažovali o spojité vlnkové transformaci. Pro kompresi obrazu je ale vhodnější diskrétní forma, neboť jsou obrazy uloženy v diskrétní paměti a prvním krokem k této formě je diskretizace kroku parametrů posunu s a dilatace r. Konkrétně $r = r_0^m$ a $s = nr_0s_0^m$ pro $r_0 > 1, s_0 > 0$ a $m \in \mathbb{Z}, n \in \mathbb{Z}$. Pokud je jádro transformace tímto způsobem diskretizováno, tak jej označujeme jako *vlnkové rámce*. Transformace je stále obecně redundantní, ale lze odstranit použitím vhodných kroků parametrů, jako je například dyadické vzorkování $r_0 = 2, s_0 = 1$ poté $r = 2^m, s = n2^m$ [21]. Dosazením dyadických parametrů do rovnice vlnky (3.2) a vlnkové transformace (3.1) dostaneme vztah Dyadické vlnkové transformace (3.9), kde m je frekvenční měřítko a n je časové posunutí. Stejně lze postupovat i u jiných transformací, které podvzorkovávají dvěma. U transformací s větším podvzorkováním se postupuje obdobným způsobem.

$$\mathrm{W}f(m,n) = \frac{1}{\sqrt{2^m}} \int f(t)\psi^* \left(\frac{t - n2^m}{2^m} \right) \mathrm{d}t \tag{3.9}$$

Celý prostor spojitých funkcí $L^2(\mathbb{R})$ lze rozložit do podprostorů V_j, rovnice (3.10). Báze podprostoru V_j je poté tvořena $\{\phi_{j,n}\}_{n\in\mathbb{Z}}$, jeho ortogonální doplněk W_j je poté definován rovnicí (3.11) a bázi tohoto podprostoru odpovídá $\{\psi_{j,n}\}_n \in \mathbb{Z}$. Hierarchie prostoru spojitých funkcí poté lze popsat pomocí doplňků rovnicí (3.12).

$$\{0\} \cdots \subset V_2 \subset V_1 \subset V_0 \subset V_{-1} \subset V_{-2} \subset \ldots L^2(\mathbb{R}) \tag{3.10}$$

$$V_{j-1} = V_j + W_j \tag{3.11}$$

$$L^2(\mathbb{R}) = \cdots \oplus W_2 \oplus W_1 \oplus W_0 \oplus W_{-1} \oplus W_{-2} \oplus \ldots \tag{3.12}$$

Svázání těchto dvou sad podprostorů podle rovnice (3.11) můžeme vyjádřit pomocí tzv. dilatačních rovnic (3.13) a (3.14). V rovnicích se vyskytují FIR filtry dolní propust $h(n)$

a horní propust $g(n)$, které svazují sousední podprostory. Dále je v rovnicích zakomponováno podvzorkování $\phi(2t-n)$. Tyto rovnice využijeme pro rozklad vstupního signálu na různé úrovně detailu. Přibližné koeficienty z prostoru V_r odpovídají poté $c_s(r) = \langle f, \phi_{r,s} \rangle$ a podrobné koeficienty odpovídají projekci do prostoru W_r vztahem $d_s(r) = \langle f, \psi_{r,s} \rangle$.

$$\phi(t) = \sqrt{2} \sum_n h(n)\phi(2t-n) \tag{3.13}$$

$$\psi(t) = \sqrt{2} \sum_n g(n)\phi(2t-n) \tag{3.14}$$

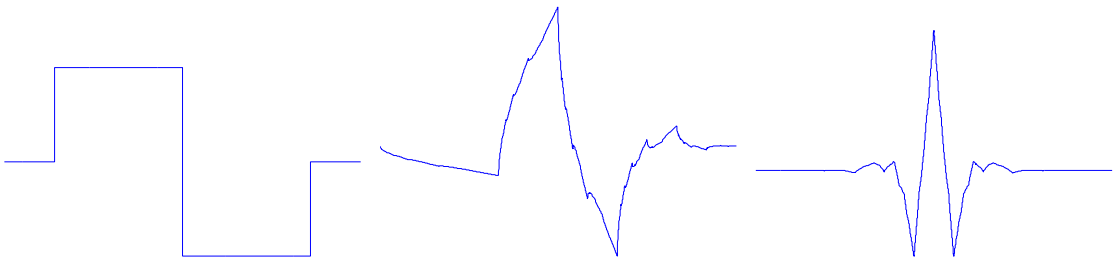
3.1 Diskrétní vlnková transformace

Dyadická vlnková transformace (3.9) se také nazývá diskrétní vlnková transformace se spojitým časem. Pokud se přesuneme na diskrétní signály, budeme potřebovat diskrétní vlnkovou transformaci s diskrétním časem. Tuto transformaci získáme použitím měřítkového a vlnkového filtru, obrázek 3.3. Výstupy těchto dvou filtrů můžeme zapsat jako (3.15) a (3.16) [18].

$$a_{j+1}[p] = \sum_{n=-\infty}^{+\infty} h[n-2p]a_j[n] \tag{3.15}$$

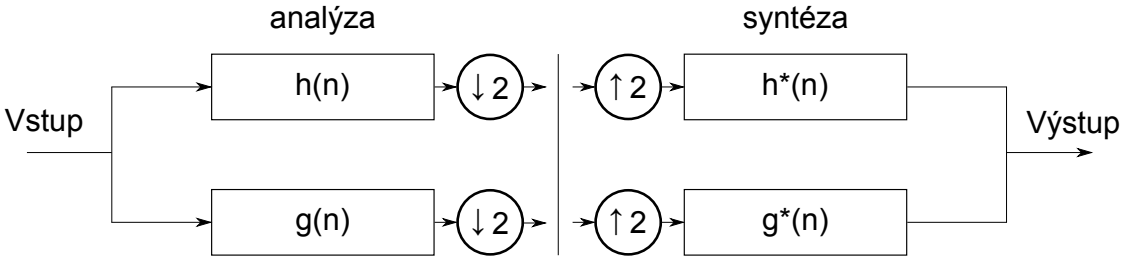
$$d_{j+1}[p] = \sum_{n=-\infty}^{+\infty} g[n-2p]a_j[n] \tag{3.16}$$

Jak je z rovnic (3.15) a (3.16) vidět, tak vstup dalšího stupně je výstup z měřítkového (dolní propusti) filtru. Výstup vlnkového (horní propusti) filtru udává detailní informace o signálu na dané úrovni. Diskrétní vlnková transformace zde popsaná je na základě konvoluce s filtry, dalším způsobem je *lifting*, viz webové stránky [32].



(a) Haarova vlnka (b) vlnka Daubechies druhé úrovně (c) Cohen-Daubechies-Feauveau vlnka 9/7

Obrázek 3.2: Příklady vlnek.



Obrázek 3.3: Blokové schéma kvadraturně zrcadlových filtrů.

Biortogonální vlnky

Sada filtrů je určena podle použité vlnky. Pro kompresi se nejčastěji používají vlnky biortogonální Cohen-Daubechies-Feauveau (CDF) dle článku [8]. Konkrétně varianty CDF 9/7 pro ztrátovou a CDF 5/3 pro bezeztrátovou kompresi. Čísla v předchozích názvech udávají velikosti filtrů. Existuje také označení vlnek biortogonální 4.4, resp. biortogonální 2.2, reprezentující tytéž vlnky, kde daná čísla určují počty nulových momentů. Vlnka CDF 9/7 je znázorněna na obrázku 3.2(c). Mezi významné vlastnosti těchto vlnek patří symetrie a velký počet nulových momentů. Symetrie vlnek odstraňuje různé artefakty v obraze a velký počet nulových momentů zlepšuje kompresi zahazováním nepotřebných koeficientů.

3.1.1 Vlnková kaskáda filtrů

Dříve zmíněné filtry $h(n)$ a $g(n)$ jsou také označovány jako měřítkový (škálovací) a vlnkový filtr. Měřítkový filtr $h(n)$ propustí nižší frekvence, které obsahují typicky vyšší energii, a reprezentují přibližné informace o signálu. Oproti tomu vlnkový filtr $g(n)$ propustí vyšší frekvence obsahující menší energii reprezentující detailní informace o signálu.

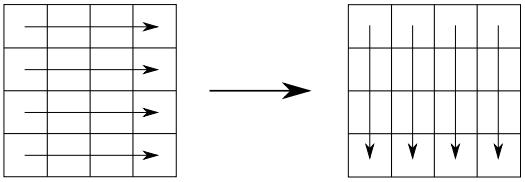
Pokud k těmto filtrům existují zrcadlové filtry $h^*(n)$ a $g^*(n)$ pro rekonstrukci, tak tuto čtveřici filtrů nazýváme *kvadraturně zrcadlové filtry* (angl. quadrature mirror filters). Blokové schéma analýzy a syntézy kvadraturně zrcadlových filtrů je na obrázku 3.3.

Kvadraturně zrcadlové filtry představují jeden stupeň analýzy a syntézy. Pokud bychom zapojily těchto filtrů více za sebe (sériově) na výstupy měřítkových filtrů, potom dostáváme tzv. kaskádu vlnkových filtrů (angl. wavelet cascade filter).

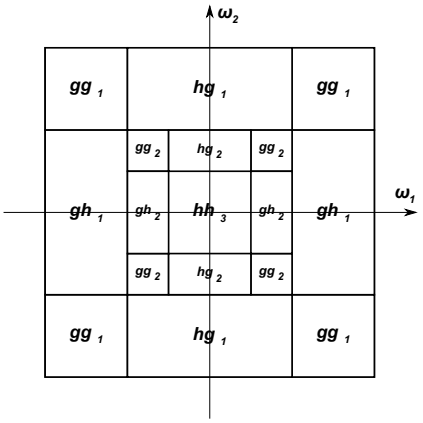
3.1.2 Diskrétní vlnková transformace ve 2D

Doposud popsaná vlnková transformace byla pro jednorozměrné signály. Pro kompresi obrazu je ale potřeba dvourozměrná transformace. Stejně jako u kosinové nebo Fourierovy transformace lze použít tzv. separabilní transformaci, kde se transformace použije na řádky a následně na sloupce, jak je znázorněno na obrázku 3.4.

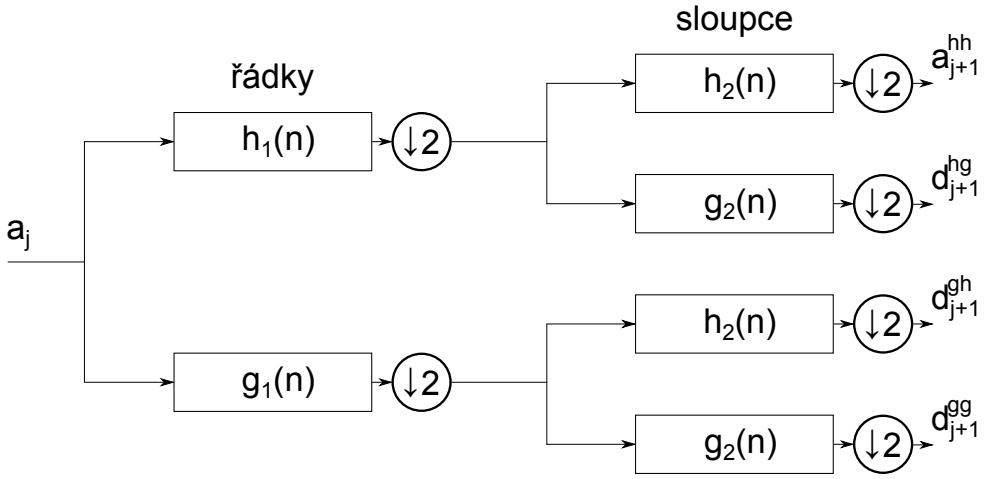
V blokovém schématu to vychází tak, že v prvním kroku se provede filtrace řádků a v druhém kroku se oba výstupy posílají na další měřítkové a vlnkové filtry podle obrázku 3.6. Lze se na tuto problematiku dívat i z pohledu, že máme čtyři filtry, které vzniknou kombinacemi měřítkového a vlnkového filtru dle rovnic (3.17). Syntéza se poté provádí pomocí druhých dvou kvadraturně zrcadlových filtrů.



Obrázek 3.4: Separabilní transformace.



Obrázek 3.5: Rozdělení spektra obrazu vlnkovou transformací.



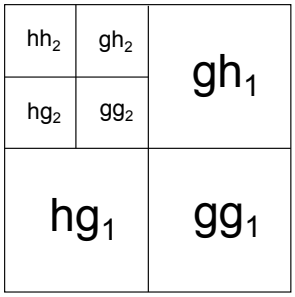
Obrázek 3.6: Blokové schéma separabilní transformace.

$$\begin{aligned} \phi(n_1,n_2) &= \phi(n_1)\phi(n_2) \\ \psi^H(n_1,n_2) &= \psi(n_1)\phi(n_2) \\ \psi^V(n_1,n_2) &= \phi(n_1)\psi(n_2) \\ \psi^D(n_1,n_2) &= \psi(n_1)\psi(n_2) \end{aligned} \tag{3.17}$$

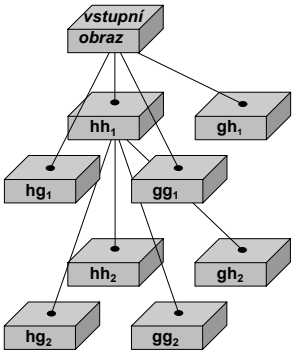
3.1.3 Koeficienty vlnkové transformace

Vlnková transformace zjednodušeně řečeno převádí informace o obraze z pixelů na vlnkové koeficienty. Výsledné koeficienty jsou logicky uspořádány do jednotlivých úrovních, kde každá úroveň představuje jeden stupeň rozkladu (viz obrázek 3.8). Tyto koeficienty se poté například kvantizují nebo jiným způsobem komprimují. Vlastní uložení koeficientů může mít vliv na výsledné vlastnosti kompresního formátu, jako je například typ progresivního přenosu (pozice vs. kvalita). Ve formátech používajících vlnkovou transformace jsou koefi-

cienty uloženy do dlaždic v uspořádání podle obrázku 3.7. Zobrazeny jsou pouze první dvě úrovně koeficientů, případné další úrovně by nahradily oblast hh_2.



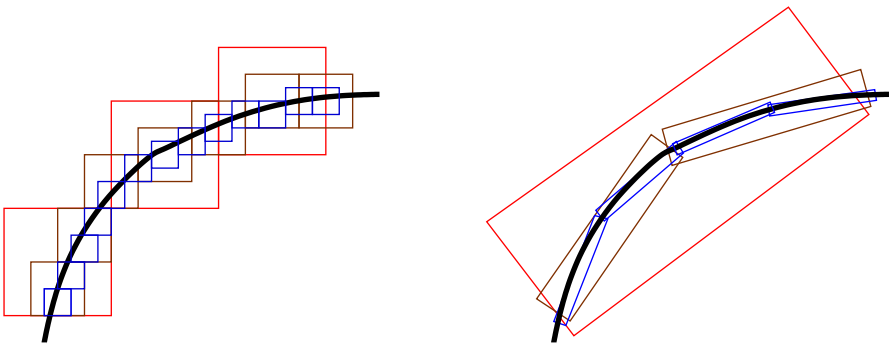
Obrázek 3.7: Organizace koeficientů vlnkové transformace.



Obrázek 3.8: První dvě úrovně rozkladu.

3.2 Konturletová transformace

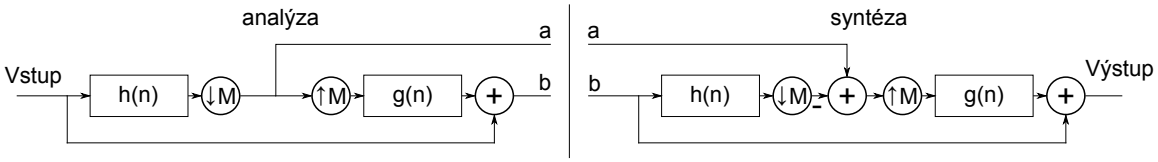
Vlnková transformace je silným nástrojem pro zpracování signálů a obrazů, jako například pro kompresi, odstranění šumu nebo detekci hran. Vlnky jsou velice dobrým nástrojem pro jednorozměrné signály, avšak jejich varianta ve dvou rozměrech už není zcela ideální. Dobrým řešením pro dvourozměrné signály (typicky obrazy) zahrnující vlastnosti, jako je směrovost a multirozklad, se ukázaly být například pro spojitý prostor kurvlety [5] a pro diskrétní konturlety [9]. V dvourozměrném obrazu vlnky zachytí hrany, ale většinou méně efektivním způsobem než konturlety, neboť konturlety „uvidí“ křivky, kdežto vlnky vidí „jen“ hrany, jak je znázorněno na obrázku 3.9.



Obrázek 3.9: Zachycení hran vlnkami a konturlety.

3.2.1 Laplaceova pyramida

Na rozdíl od vlnkové transformace je pro multirozklad konturletové transformace použita Laplaceova pyramida (dále jen LP) dle článku [4]. Rozklad pomocí LP generuje na každé úrovni podvzorkovanou nízkofrekvenční verzi původního obrázku (obecně signálu) a rozdíly mezi původním. Dekompozice je znázorněna na obrázku 3.10, kde $h(n)$ a $g(n)$ jsou filtry pro



Obrázek 3.10: Analýza a syntéza signálu pomocí Laplaceovi pyramidy.

analýzu a syntézu. Filtr $g(n)$ definuje unikátní škálovací funkci (3.18), podobně filtr $h(n)$ definuje funkci (3.19). Výstupy jsou obraz a pro hrubou podvzorkovanou nízkofrekvenční verzi původního signálu a obraz b jsou diference mezi rekonstruovanou hrubou a původní verzí. Výstup z filtru $h(n)$ je podvzorkován maticí M. Aplikací matice M na vstupní signál $x[n]$ dostaneme podvzorkovaný signál $x_d[n] = x[Mn]$, kde M je celočíselná matice. LP implicitně nadvzorkuje až do výše 33 %, ale na rozdíl od vlnkové transformace generuje pouze jeden všesměrový „obraz“ namísto tří, jako je na obrázku 3.7.

$$\phi(t) = 2 \sum_{n \in \mathbb{Z}^2} g[n]\phi(2t - n) \tag{3.18}$$

$$\psi(t) = 2 \sum_{n \in \mathbb{Z}^2} h[n]\psi(2t - n) \tag{3.19}$$

3.2.2 Směrová banka filtrů

Směrová banka filtrů dle práce [22] slouží k analýze obrazu na rozklad do obrazů pro jednotlivé směry. Rozklad je tvořen binárním stromem úrovně l, který vede na 2^l podpásem trojúhelníkového tvaru (viz obrázek 3.12).

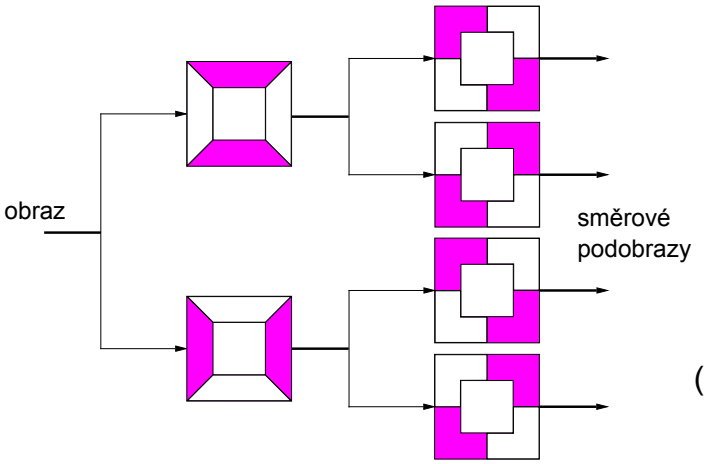
Směrová banka filtrů je složena ze dvou složek. První složkou jsou quincunx banka filtrů dle článku [34], která rozděluje 2D spektrum na horizontální a vertikální části. Druhou komponentou jsou tzv. „shearing“ operátory. Tyto operátory slouží pouze k natočení původního signálu například o 45 stupňů. Vhodnou kombinací quincunx filtrů a „shearing“ operátorů můžeme vytvořit směrovou banku filtrů, viz obrázek 3.11, pro rozdělení 2D spektra. Zpětná rekonstrukce je tvořena inverzními operacemi.

Na rozklad úrovně l se lze také dívat jako na banku 2^l paralelních filtrů se vzorkovacími maticemi $\mathbf{S}$. Jednotlivé směrové filtry $D_k^{(l)}, 0 \leq k < 2^l$ odpovídají jednotlivým pásmům v obrázku 3.12. Báze diskrétních signálů v $l^2(\mathbb{Z}^2)$ je poté reprezentována v (3.20). Z rovnice (3.21) lze vyvodit separabilitu vzorkování. Tyto dvě sady reprezentují víceméně horizontální a vertikální směry.

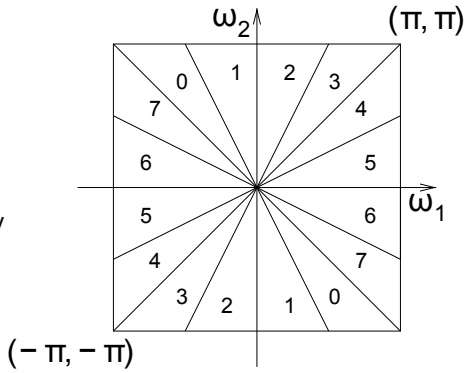
$$\left\{ d_k^{(l)}[n - S_k^{(l)} m] \right\}_{0 \leq k < 2^l, m \in \mathbb{Z}^2} \tag{3.20}$$

$$\mathbf{S}_k^{(l)} = \begin{cases} \operatorname{diag}(2^{l-1}, 2) & \text{pro } 0 \leq k < 2^{l-1}, \\ \operatorname{diag}(2, 2^{l-1}) & \text{pro } 2^{l-1} \leq k < 2^l \end{cases} \tag{3.21}$$

Na obrázku 3.12 je zobrazena směrová banka filtrů 3. úrovně obsahující 2^3 směrových podpásem.



Obrázek 3.11: Schéma konturletové směrové banky filtrů.



Obrázek 3.12: Rozdělení spektra na 8 směrových podpásem.

3.2.3 Konturletová banka filtrů

Směrová banka filtrů dobře zachycuje vyšší frekvence (dále od středu), jak je zobrazeno na obrázku 3.12, ale nižší frekvence se projeví ve více směrových pásmech. Tohoto nedostatku se lze zbavit zkombinováním směrové banky filtrů s multirozkladem pomocí Laplaceovy pyramidy. LP efektivně oddělí nižší frekvence od vyšších, na které poté lze aplikovat směrovou banku filtrů, jak je zobrazeno na obrázku 3.13. Tuto kombinaci LP a směrové banky filtrů budeme nazývat *konturletovou bankou filtrů*. Rozklad obrazu probíhá následovně. Vstupní obraz $a_0[n]$ je rozložen LP na J pásmových obrazů $b_j[n], j = 1, 2, \ldots, J$ a nízkofrekvenční obraz $a_j[n]$. Každý pásmový obraz $b_j[n]$ je přitom dále rozložen směrovou bankou filtrů úrovně l_j na 2^{l_j} směrových (pásmových) obrazů $c_{j,k}^{(l_j)}[n], k = 0, 1, \ldots, 2^{l_j} - 1$.

Využitím $\psi(t)$ z (3.19) je vytvořen těsný frame, rovnice (3.22), pro obraz detailů (vysoké frekvence) na dané úrovni j.

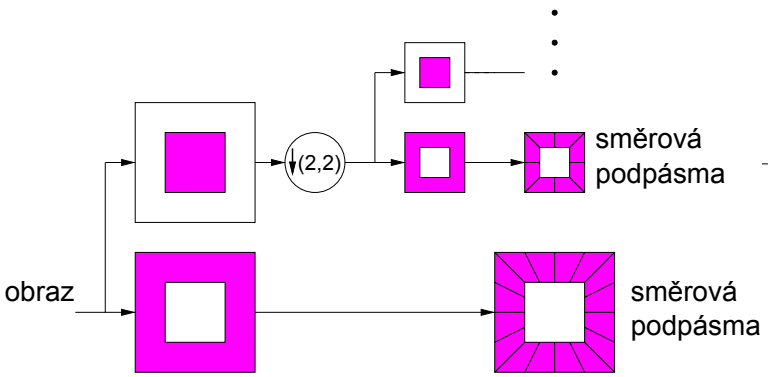
$$\psi_{j,n}(t) = 2^{-j}\psi\left(\frac{t - 2^j n}{2^j}\right), \quad j \in \mathbb{Z}^2 \tag{3.22}$$

Takto vytvořený těsný frame není invariantním podprostorem k posunu. Tento nedostatek se odstraní rovnicí (3.23). V rovnici (3.23) k_i označuje coset pro podvzorkování dvěma v každé dimenzi.

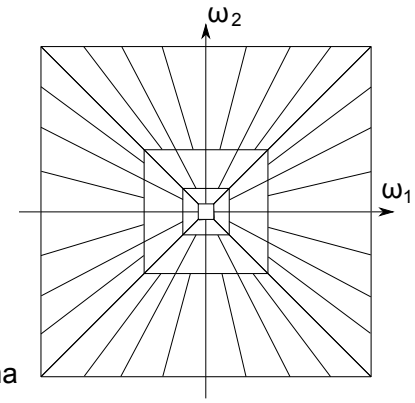
$$\mu_{j,2n+k_i}(t) = \psi_{j,n}(t) \tag{3.23}$$

Těsný frame pro směrová podpásma obrazů detailů je dán (3.20) a (3.23) spojenými v rovnici (3.24). Indexy j, k a n označují měřítko, směr a lokaci.

$$\lambda_{j,k,n}^{(l)}(t) = \sum_{m \in \mathbb{Z}^2} d_k^{(l)}[m - S_k^{(l)} n]\mu_{j,m}(t) \tag{3.24}$$



Obrázek 3.13: Konturletová banka filtrů složená z LP a směrové banky filtrů.



Obrázek 3.14: Rozdělení spektra na měřítka a směrová podpásma.

Počet úrovní rozkladu směrové banky filtrů l může být jiný na každé úrovni j rozkladu LP (viz obrázek 3.14), proto bude označován jako l_j. Konturletová transformace je poté dána rovnicemi (3.25) a (3.26), kde $f(t) \in L_2(\mathbb{R}^2$.

$$a_j[n] = \langle f, \phi_{L+j,n} \rangle \tag{3.25}$$

$$c_{j,k}^{(l_j)}[n] = \langle f, \lambda_{L+j,k,n}^{(l_j)} \rangle \tag{3.26}$$

Konturletová transformace provedená takto vytvořenou konturletovou bankou filtrů (její schéma je zobrazeno na obrázku 3.13) má následující vlastnosti:

- Pokud LP i směrová banka filtrů obsahuje filtry dokonalé rekonstrukce, pak i diskrétní konturletová transformace dosahuje dokonalé rekonstrukce.
- Pokud LP i směrová banka filtrů obsahuje ortogonální filtry, pak diskrétní konturletová transformace zajistí těsný frame s mezemi rovny 1.
- Diskrétní konturletová transformace obsahuje redundanci menší než 33 %.
- V případě směrové banky filtrů úrovně l_j použitou na LP úrovně j, poté bázové obrazy diskrétní konturletové transformace mají šířku $\approx C2^j$ a délku $\approx C2^{j+l_j-2}$.
- Použitím FIR filtrů je výpočetní náročnost konturletové transformace třídy $O(N)$ pro obraz o velikosti N pixelů.

Důkazy těchto tvrzení jsou k nalezení v [9].

3.3 Shearletová transformace

Shearletová transformace [10] staví na speciální podskupině vlnek zvaných shearlety, které jsou dány následujícím vztahem (3.27) ve spojitém prostoru.

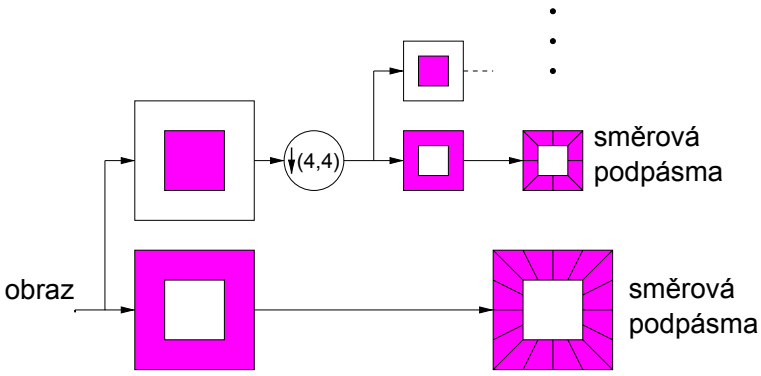
$$\psi_{a,s,t}(x) = a^{-3/4}\psi(D_{a,s}^{-1}(x-t)), \quad \text{kde } D_{a,s} = [a, -a^{1/2}s; 0, a^{1/2}] \tag{3.27}$$

Mateřská shearlet funkce ψ je definována tenzorovým součinem (3.28), kde ψ_1 je vlnka a ψ_2 je bump funkce.

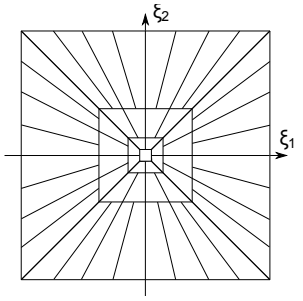
$$\psi(\xi) = \psi(\xi_1, \xi_2) = \psi_1(\xi_1)\psi_2(\xi_2/\xi_1) \tag{3.28}$$

Shearletová transformace ve spojitém prostoru je vyjádřena vztahem (3.29).

$$\mathrm{SH}f(a,s,t) = \langle f, \psi_{a,s,t}\rangle \tag{3.29}$$



Obrázek 3.15: Kombinace Laplaceovy pyramidy a směrových filtrů u shearletové transformace. Pouze dvě úrovně rozkladu.



Obrázek 3.16: Rozdělení spektra na měřítka a směrová podpásma.

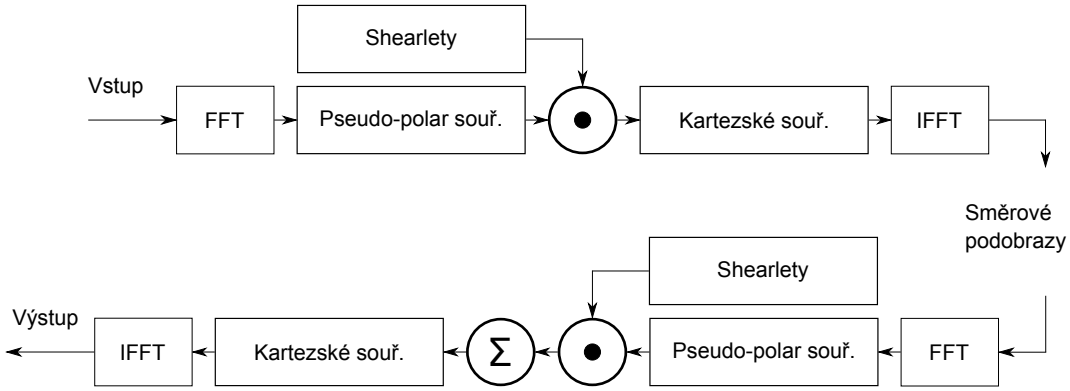
3.3.1 Diskrétní shearletová transformace

Diskrétní shearletová transformace je dosažena vhodným vzorkováním její spojité varianty. Je tedy použita příslušná diskrétní množina měřítkových, shear a translačních parametrů. Jako mateřská vlnka ψ_1 se použije diskrétní vlnka. Koeficienty $\langle f, \psi_{j,l,k}^{(d)}\rangle$ jsou dány rovnicí (3.30).

$$\langle f, \psi_{j,l,k}^{(d)}\rangle = 2^{\frac{3j}{2}} \int_{\mathbb{R}^2} \hat{f}(\xi)\overline{V(2^{-2j}\xi)W_{j,l}^{(d)}(\xi)}e^{2\pi i \xi A_d^{-j} B_d^{-l} k} d\xi \tag{3.30}$$

V a $W_{j,l}^{(d)}$ jsou okénkové funkce pracující podle obrázku 3.16. $\hat{f}(\xi)$ značí 2D Fourierovu transformaci. Dále $d = \{0, 1\}$ a matice A_d a B_d jsou dány v (3.31).

$$A_0 = \begin{pmatrix} 4 & 0 \\ 0 & 2 \end{pmatrix}, \quad B_0 = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix}, \quad A_1 = \begin{pmatrix} 2 & 0 \\ 0 & 4 \end{pmatrix}, \quad B_1 = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \tag{3.31}$$



Obrázek 3.17: Blokové schéma analýzy a syntézy směrové banky filtrů shearletové transformace.

Diskrétní shearletová transformace je v blokovém schématu na obrázku 3.15 podobná konturletové transformaci s rozdílem ve větším podvzorkováním. Rozložení na nízkofrekvenční a vysokofrekvenční obraz je vytvořeno například Laplaceovou pyramidou, viz schéma na obrázku 3.10. Schéma směrové banky filtrů je zobrazeno na obrázku 3.17. Pro získání rozložení koeficientů podle obrázku 3.16 se aplikuje na vstupní signál (obraz) diskrétní Fourierova transformace a výsledné koeficienty se přeuspořádají do pseudo–polární mřížky. Převod na pseudo–polární souřadnice je proveden podle rovnic (3.32), kde $\mathcal{D}_0$, resp. $\mathcal{D}_1$, reprezentuje horizontální, resp. vertikální, výřez.

$$\begin{aligned}(u,v) &= (\xi_1, \frac{\xi_2}{\xi_1}) \text{ pokud } (\xi_1,\xi_2) \in \mathcal{D}_0 \\ (u,v) &= (\xi_2, \frac{\xi_1}{\xi_2}) \text{ pokud } (\xi_1,\xi_2) \in \mathcal{D}_1\end{aligned} \tag{3.32}$$

Aplikace změny souřadnic je vyjádřena jako $g_j(u,v) = \hat{f}_d^j(\xi_1,\xi_2)$ a pro $l = 1-2^j, \cdots, 2^j - 1$ dostaneme (3.33).

$$\hat{f}(\xi)\overline{V(2^{-2j}\xi)W_{j,l}^{(d)}(\xi)} = g_j(u,v)\overline{W(2^j v - l)} \tag{3.33}$$

Koeficienty shearletové transformace jsou tedy dány rovnicí (3.34).

$$\langle f, \psi_{j,l,k}^{(0)} \rangle = \int\int 2^{\frac{3j}{2}} g_j(u,v)\overline{W(2^j v - l)}\exp\left(2\pi i\left(\frac{n_1 + ln_2}{4^j}\xi_1 + \frac{n_2}{2^j}\xi_2\right)\right) d\xi_1 d\xi_2 \tag{3.34}$$

Pokud $\{w_{j,l}[n] : n \in \mathbb{Z}\}$ je sekvence, jejíž diskrétní Fourierova transformace dává diskrétní vzorky okénkové funkce W, tak platí $\hat{w}_{j,l}[k] = \overline{W(2^j k - l)}$. Potom aplikace okénkové funkce je dána součinem (3.35).

$$g_j[n_1,n_2]\hat{w}_{j,l}[k] \tag{3.35}$$

Krok shearletové transformace na úrovni j lze popsat jednoduše několika kroky.

- Aplikace Laplaceovy pyramidy pro dekompozici f_d^{j-1} na nízkofrekvenční obraz f_a^j a vysokofrekvenční obraz f_d^j. Pro $f_d^{j-1} \in l^2(\mathbb{Z}^2_{N_j-1})$, matice $f_a^j \in l^2(\mathbb{Z}^2_{N_j})$, kde $N_j = N_{j-1}/4$ a $f_d^j \in l^2(\mathbb{Z}^2_{N_j})$.
- Vypočítat $\hat{f}_d^j$ na pseudo–polárních souřadnicích. Tímto je získaná matice Pf_d^j.
- Aplikace pásmových filtrů podle (3.35).
- Zpětný převod do kartézských souřadnic a aplikace inverzní dvoudimenzionální Fourierovy transformace.

3.4 Komprese koeficientů

Koeficienty transformací jsou logicky uskupeny do stromovitých struktur. Podobně jako u kosinové transformace (formát JPEG) můžeme výsledné koeficienty kvantizovat a uchovat například pouze **N** nejvýznamnějších koeficientů. Z toho plyne, že lze pro kódování koeficientů použít „vložené“ kódování (Embedded coding) jak je tomu u vlnkové transformace. Obecně se embedded kódování skládá z následujících kroků podle knihy [21]:

1. *Inicializace*: uložení indexu n prvního neprázdného setu koeficientů Θ_n
2. *Kódování významnosti*: Uložení mapy významnosti $b_n[m]$ pro $m \notin \Theta_{n+1}$
3. *Kódování znaménka*: zakódování znaménka pro koeficienty $f_{\mathcal{B}}[m]$ pro $m \in \Theta_n$
4. *Zjemnění kvantizace*: upřesnění koeficientů splňující podmínku $|f_{\mathcal{B}}[m]| > 2^{n+1}$
5. *Zjemnění přesnosti*: snížení n o 1 a skok na 2.

Algoritmus lze zastavit v jakémkoliv bodě, neboť jsou kódovány od hrubých po jemné. Místo ukončení poté udává kvalitu/velikost výsledného obrazu.

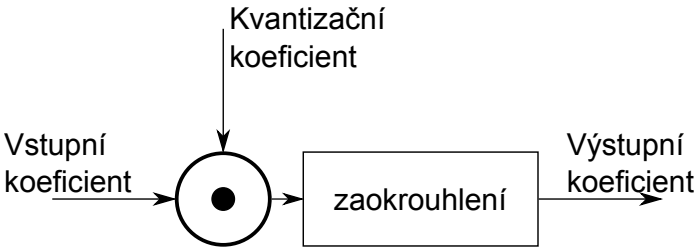
Kvantizace

Výstupem jednotlivých transformací je posloupnost reálných čísel Algoritmy pro kompresi koeficientů pracují většinou s celými čísly. Proces, jenž provádí tento převod z reálných do celých čísel se nazývá kvantizace. Tento proces je prakticky vždy nenávratně ztrátový. Přesnost kvantovaných hodnot udává počet kvantovacích hladin, tj. počet a rozptyl možných hodnot, kterých mohou koeficienty nabývat. Nejjednodušší způsob kvantizace je prosté zaokrouhlení reálných čísel na nejbližší celá čísla. Počet kvantizačních hladin je poté rozdíl největší a nejmenší kvantované hodnoty.

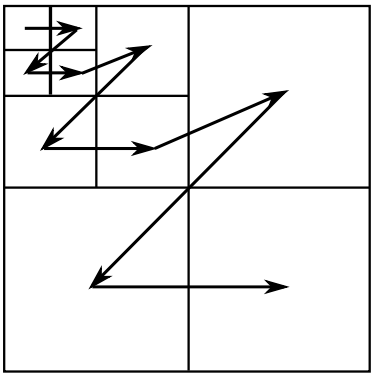
Kvantizace lze uzpůsobit různým potřebám. Lze například před samotnou kvantizací násobit koeficienty konstantou pro zmenšení nebo zvětšení počtu kvantovacích hladin a tedy ovlivnit hrubost kvantizace. Dále je možné rozprostřít kvantizační úrovně do daného intervalu pro lepší zpracování po bitových rovinách.

3.4.1 EZW

Emmbedded zerotree wavelet dle článku [27] patří mezi nejzákladnější kódování koeficientů diskrétní vlnkové transformace. Často je dále různě upravován pro lepší efektivitu (například



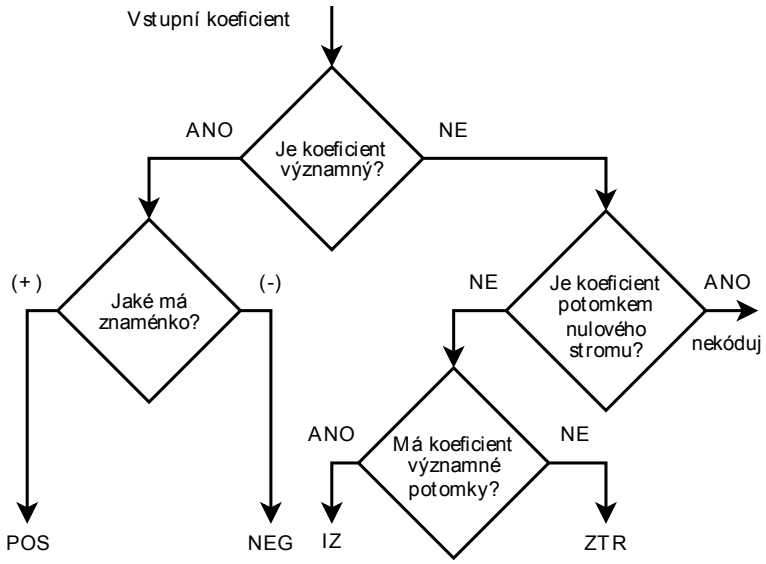
Obrázek 3.18: Blokové schéma kvantizace.



Obrázek 3.19: EZW průchod mezi třemi úrovněmi koeficientů.

algoritmus SPIHT). Mezi hlavní výhody patří možnost postupného přenosu s průběžným zpřesňováním s možností přerušení toku po dosažení určitých podmínek.

Jednotlivé koeficienty jsou u transformací řazeny podle jejich důležitosti a tento fakt je dále využit při aplikaci *EZW* jak je například zobrazeno na obrázku 3.19. Základem algoritmu je nalezení největšího koeficientu, od kterého následně závisí počet iterací. V každé iteraci se bude posupně snižovat práh, jenž původně vychází z největšího koeficientu.



Obrázek 3.20: Rozhodování o symbolu v dominantním průchodu EZW.

Každá iterace začíná dominantním průchodem, viz obrázek 3.20, který zakóduje koeficienty podle jejich typu na základě aktuálního prahu. Existují čtyři typy. **POS** a **NEG** reprezentují koeficient, který se bude dále kódovat v podřízeném průchodu. **ZTR** slouží k označení kořene nulového stromu, jenž říká, že všichni jeho potomci jsou nedůležití. Zde je největší komprese algoritmu. **IZ** značí izolovanou nulu, neboli nevýznamný koeficient, který má ovšem ve svém podstromu nějaké významné koeficienty.

Druhá část iterace (podřízený průchod) spočívá v kódování samotných koeficientů na základě mapy významnosti složené v předchozím kroku. Vysoká časová náročnost je dána

výpočtem pro rozlišení mezi **ZTR** a **IZ** pro potenciálně každý koeficient. Výstupem dominantního průchodu je seznam čtyř znaků **POS**, **NEG**, **ZTR** a **IZ** a výstupem podřízeného průchodu jsou hodnoty bitů zpracovaných prvků (z dominantního průchodu) podle aktuálního prahu. Na celý výstup EZW je dále aplikován aritmetický kodér.

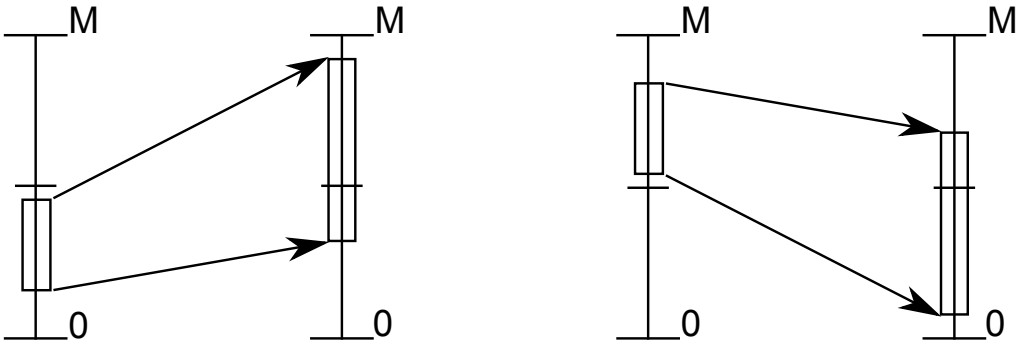
Algorithm 1 EZW algoritmus

(Inicializace)
$threshold := 2^{\log_2(MAXcoef)}$
(Dominantní průchod)
Inicializace fronty FIFO
while neprázdné FIFO **do**
  získej kódový symbol prvku z FIFO;
  if symbol je POS, NEG nebo IZ **then**
    zpracuj potomky daného prvku a ulož je do fronty FIFO;
    if symbol je POS nebo NEG **then**
      ulož prvek bez znaménka do seznamu LIST;
      vynuluj zpracovávaný prvek;
    end if
  end if
end while
(Podřízený průchod)
$threshold_{sub} := threshold/2$;
for všechny prvky v seznamu LIST **do**
  if $koeficient > threshold_{sub}$ **then**
    odešli 1;
    $koeficient := koeficient - threshold_{sub}$;
  else
    odešli 0;
  end if
end for
(Další krok)
$threshold := threshold/2$;
if $threshold > threshold_{min}$ **then**
  go to (Dominantní průchod);
end if

Aritmetický kodér

Zakódované koeficienty některými algoritmy (například EZW) je vhodné ještě jednou zakódovat aritmetickým kodérem dle [37]. Je to dáno omezeným množstvím výstupních symbolů, které se opakují s určitou pravděpodobností.

Aritmetický kodér začíná na intervalu $\langle 0, 1)$ reálných čísel, který rozdělí na podintervaly o velikosti přímo úměrné pravděpodobnosti výskytu daných symbolů. Z toho vyplývá nutná znalost počtu výskytů jednotlivých symbolů. Běžně je používána adaptivní varianta, která na začátku přiřadí všem symbolům pevně danou pravděpodobnost a tabulku výskytů po přečtení určitého počtu symbolů aktualizuje. Tímto odpadá nutnost přenášet tabulku pravděpodobností pro dekódování. Jediný potřebný údaj pro dekódování, za předpokladu



Obrázek 3.21: Příklad rozšíření intervalu binárního Aritmetického kodéru shora a zdola.

známých výchozích hodnot pravděpodobností, je počet zakódovaných symbolů.

Po zakódování posledního symbolu je vybraná nejkratší hodnota z aktuálního intervalu. Tato reálná hodnota reprezentuje celý řetězec zakódovaných symbolů a k jejímu dekódování je potřeba pouze počet zakódovaných symbolů, pokud nebyl do toku přidán symbol pro konec řetězce.

Popsaná verze pracuje s reálnými čísly. Tato varianta není moc používaná, neboť má potenciálně vysoké nároky na přesnost čísel. Často je tedy používaná celočíselná varianta pracující na bitovém toku. Princip je pořád stejný, ale pracuje se s intervalem celých čísel v rozsahu $\langle 0, M)$, kde M je dostatečně velké celé číslo. Určení horní hranice je často závislé na použitém datovém typu. Například při použití 32bitového čísla je použito $M = 2^{31} - 1$, vychází se z největšího kladného čísla s rezervou proti přetečení, neboť při výpočtu může dojít k překročení horní hranice.

Místo pravděpodobností (desetinná čísla) je možné použít četnosti výskytu. Tyto četnosti mohou narůstat až do přetečení datového typu. Proto se určí práh, který při překročení způsobí zkrácení všech četností větších jak 1 na polovinu.

Zakódování jednoho znaku znamená posunutí hranic intervalu. Toto zmenšování při práci s celými čísly časem způsobí nemožnost korektně rozdělit daný interval na podintervaly. Je tedy určena hranice (zpravidla polovina intervalu), která signalizuje nutnost interval roztáhnout na vhodnější rozsah. Na obrázku 3.21 jsou uvedeny obě varianty roztažení. Při roztažení intervalu je na výstup odeslána hodnota indikující z které strany byla překročena hranice.

3.4.2 SPIHT

Algoritmus *Set Partitioning in Hierarchical Trees* je navržen pro koeficienty vzniklé pyramidovým rozkladem. SPIHT vychází z předchozího algoritmu EZW. Podobně jako EZW využívá vlastností rozkladu, jako je například uspořádané bitové roviny a korelace koeficientů mezi jednotlivými úrovněmi.

SPIHT dle článku [25] provádí *progresive transmition scheme*, jenž odpovídá embedded kódování. Vliv na snížení odlišností v obraze přímo určuje prioritu daného bitu. To znamená, že jako první jsou zpracovány koeficienty s největší amplitudou, neboť vedou k největšímu poklesu odlišnosti. Mezi významné vlastnosti, a také odlišnosti od EZW, patří systém kódování, jenž explicitně nepřenáší informace o pořadí kódovaných bitů. Pro dekódování je ovšem pořadí kódovaných bitů nezbytné. SPIHT využívá informaci o „vykonávací

cestě“ (z angl. *execution path*). Vykonávací cesta je seznam rozhodnutí na všech větveních v průběhu kódování. Pokud dekodér zná tuto vykonávací cestu, je poté schopen odvodit pořadí zakódovaných bitů. Absence pořadové informace má poté za důsledek zbytečnost symbolů, které byly v EZW. Algoritmus odstraňuje natolik redundanci, že přínos aritmetického kodéru je tak malý, viz srovnání v článku [23], oproti výpočetní náročnosti, že se od něj zpravidla upouští. Výstup algoritmu SPIHT je čistě bitový.

Algoritmus SPIHT pracuje s prostorovými stromy, které jsou vytvořeny předcházející transformací (např. vlnkovou). Jedná se o prostorovou korelaci jednotlivých koeficientů na různých úrovních rozkladu (viz obrázek 3.22. Každý bod v rozkladu je reprezentován jedním uzlem. Jeho přímí potomci jsou všechny body o úroveň výše v jeho prostorové lokalitě.

Algoritmus SPIHT definuje čtyři množiny koeficientů:

- $\mathcal{O}(i,j)$ je množina všech přímých potomků uzlu (i,j)
- $\mathcal{D}(i,j)$ je množina všech potomků uzlu (i,j)
- $\mathcal{H}(i,j)$ je množina všech kořenů prostorových stromů
- $\mathcal{L}(i,j)$ je množina všech nepřímých potomků uzlu (i,j)

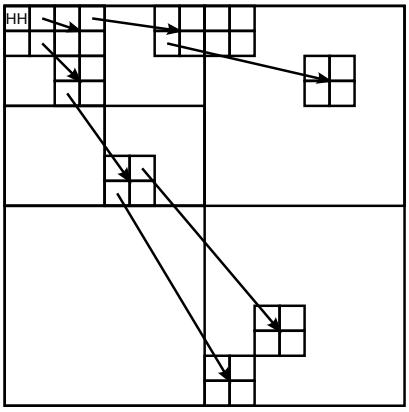
Algoritmus SPIHT uchovává informace o seřazení určitých množin koeficientů. To je realizováno pomocí seznamů nevýznamných množin *LIS*, nevýznamných bodů *LIP* a významných bodů *LSP*. Každý koeficient v seznamech je identifikován svými souřadnicemi (i,j). Seznamy *LIP* a *LSP* obsahují jednotlivé koeficienty a seznam LIS obsahuje položky z množin $\mathcal{D}$ a $\mathcal{L}$. Algoritmus SPIHT lze rozdělit na celkem čtyři části.

Inicializace je provedena pouze jednou. V tomto kroku jsou inicializovány seznamy *LIS*, *LIP*, *LSP* a kvantizační krok.

Srovnávací průchod zjišťuje změny ve významnosti koeficientů v seznamech *LIP* a *LIS*. Provádí se test položek seznamu *LIP*, zda se nestaly významnými. Pokud ano, tak jsou přesunuty do *LSP*. Položky ze seznamu *LIS* jsou rovněž testovány, jestli se nestaly významnými. V kladném případě jsou rozděleni na potomky, kteří jsou přesunuti na konec seznamu *LIS*.

Upřesňovací průchod odesílá nejvýznamnější bit, podle současného kvantizačního kroku, na výstup, pro všechny položky seznamu LSP (kromě těch přidaných v posledním srovnávacím průchodu)

Aktualizace kvantizačního kroku snižuje kvantizační krok o 1.



Obrázek 3.22: Prostorový strom vzniklý vlnkovou transformací.

Algorithm 2 SPIHT algoritmus

(Inicializace)
$n := \lfloor \texttt{log}_2(\texttt{MAX}) \rfloor$
$LSP := \psi$;
$LIP := \mathcal{H}$;
$LIS :=$ pouze ty prvky z $\mathcal{H}$, které mají přímé potomky, jako prvky typu A;
(Srovnávací průchod)
for all $(i,j) \in LIP$ **do**
    odešli $S_n(i,j)$;
    if $S_n(i,j) = 1$ **then**
        přesuň (i,j) do LSP a odešli znaménko $c_{i,j}$;
    end if
end for
for all $(i,j) \in LIS$ **do**
    if prvek je typu A **then**
        odešli $S_n(\mathcal{D}(i,j))$;
        if $S_n(\mathcal{D}(i,j)) = 1$ **then**
            for all $(k,l) \in \mathcal{O}(i,j)$ **do**
                odešli $S_n(k,l)$;
                if $S_n(k,l) = 1$ **then**
                    přidej (k,l) do LSP a odešli znaménko $c_{k,l}$;
                end if
                if $S_n(k,l) = 0$ **then**
                    přidej (k,l) na konec LIP;
                end if
            end for
            if $\mathcal{L}(i,j) \neq \psi$ **then**
                přesuň (i,j) na konec LIS jako prvek typu B;
            else
                odeber z LIS prvek (i,j);
            end if
        end if
    end if
    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;
        end if
    end if
end for
(Upřesňovací průchod)
for all $(i,j) \in LSP$ vyjma prvků přidaných v posledním srovnávacím průchodu **do**
    odešli n-tý nejvýznamnější bit $|c_{i,j}|$;
end for
(Další krok)
$n := n - 1$;
go to (Srovnávací průchod);

3.4.3 Další algoritmy

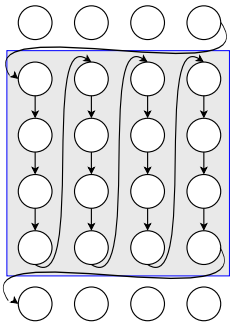
Existuje řada dalších algoritmů pro kódování koeficientů transformací. Mezi nejznámější patří mimo dříve zmíněných ještě algoritmus *EBCOT*[28], popsaný dále, který je použit v grafickém formátu JPEG 2000. Ve formátu JPEG je komprese soustředěna do bloku kvantizace a koeficienty jsou poté kódovány poměrně jednoduchým algoritmem *Run–lenght encoding* (RLE), který posloupnosti stejných symbolů nahrazuje dvojicí počtu opakování a daným symbolem. Další méně známé algoritmy jsou například *Embedded ZeroBlock Coding* (EZBC)[15], *Morphological Representation of Wavelet Data* (MRWD)[26], *Significance Linked Connected Component Analysis* (SLCCA)[6], *Wavelet Difference Reduction* (WDR)[30] a další.

EBCOT

Předchozí metody kódování koeficientů EZW a SPIHT vytváří závislosti mezi jednotlivými úrovněmi rozkladu. Tato vlastnost do určité míry omezuje možnou paralelizaci [35]. Kromě první základní úrovně, se musí dekódovat každá úroveň s pomocí předchozích vrstev.

Embedded Block Coding with Optimized Truncation (zkráceně EBCOT) se snaží nevytvářet tyto závislosti napříč úrovněmi. Namísto se koeficienty dekorelují pouze na příslušné úrovni. Případné chyby jsou izolované pouze na danou vrstvu. Tento algoritmus je použit ve formátu JPEG 2000.

Algoritmus před kódováním rozdělí vstupní koeficienty na bloky o velikosti 8×8. Každý blok je poté procházen po pruzích o výšce čtyřech koeficientů. Jednotlivé pruhy se procházejí po sloupcích. Průchod je graficky znázorněn na obrázku 3.23.



Obrázek 3.23: Průchod koeficienty blokem 4×4 v EBCOT

Při samotném kódování jsou definovány čtyři primitiva: *Run-length*, *Zero coding*, *magnitude refinement* a *Sign coding*. Koeficienty jsou předešlým způsobem uskutečněny průchody zvané Significance propagation, Magnitude refinement a Cleanup pass. V jednotlivých průchodech jsou přiřazena některá z primitiv, jenž reprezentují tabulku obsahující kontexty daného koeficientu z ostatních podpásem na stejné úrovni. Kontexty se určují pomocí sousedů v pásmech na stejné úrovni a slouží k určení uspořádání a významu v bitovém toku pro dekódování. Kóduje se kontext společně s daným bitem.

Significance propagation je průchod, ve kterém se kódují nejbližší sousedi (osmiokolí) významného koeficientu. Důvod je ten, že koeficienty s velkými hodnotami se často nacházejí ve shlucích. Použitými primitivy jsou pouze ZC a SC.

Magnitude refinement je průchod jenž zpřesňuje koeficienty označené dříve jako významné. Použito je pouze primitivum MR.

Cleanup pass provádí „úklid“ pro koeficienty nekódované v předcházejících dvou průchodech. V průchodu je použit speciální symbol UNI, jenž slouží k určení prvního významného koeficientu v daném sloupci. V daném sloupci se začne kódovat jedním RL, dokud se nenalezne první významný koeficient. Ten je označen symbolem UNI a jeho znaménko je zakódováno pomocí SC. Zbývající koeficienty ve sloupci jsou zakódovány jako ZC. Určení kontextu primitiva je provedeno porovnáním osmiokolí koeficientů s tabulkami primitiv. V první iteraci bitové roviny se provádí pouze tento průchod.

Po ukončení všech iterací je celý výstup kódován speciálním aritmetickým kodérem *MQ-coder* popsaném v článku [29]

Kapitola 4

Implementace

Tato kapitola se věnuje jednotlivým částem postupu implementace aplikace a knihovny. Cílem této práce je vytvořit knihovnu pro kompresi obrazu pomocí dříve zmíněných transformací v kombinaci s různými parametry, jako například volbou použité vlnky nebo zvoleného algoritmu pro kompresi výsledných koeficientů transformace. Výsledná knihovna bude poté implementována v jazyce C nebo C++. Součástí knihovny jsou také funkce pro zpětnou transformaci za účelem porovnání kvality obrazu při různých úrovních komprese.

4.1 Návrh

Cílem návrhu je vytvořit teoretickou kostru pro následující implementaci určující hlavní body implementace, které se v následné implementaci rozvinou. Výsledek návrhu není vázán na konkrétní implementační jazyk a proto se nebude zabývat konkrétními specifiky implementačního jazyka.

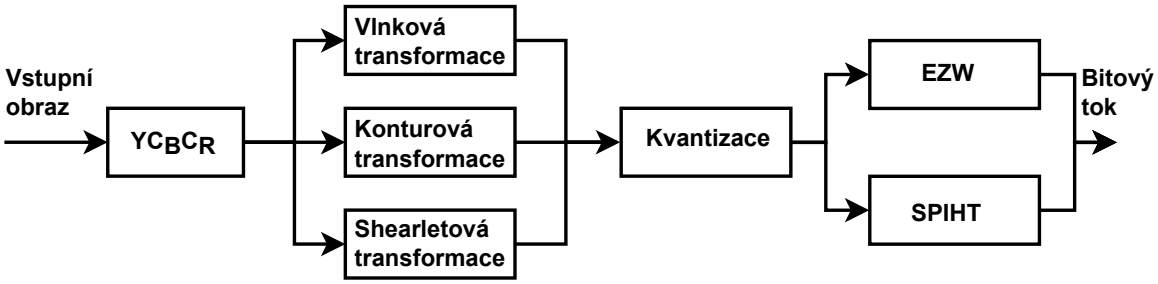
Architektura knihovny

Cílem je vytvořit knihovnu pro kompresi obrazu s několika různými parametry. Celý proces komprese lze logicky rozdělit na několik po sobě jdoucích funkčních bloků. Tento přístup umožní, aby aplikace při každém spuštění vybrala pouze požadované funkční bloky pro kompresi a umožňuje snadnou rozšiřitelnost a odstraňuje nutnost nové rekompilace pro například jiné, již implementované, transformace. Výsledná struktura aplikace tak nebude zajišťovat nejrychlejší zpracování, ale cílem této práce je porovnat efektivitu zvolené kombinace algoritmů.

Prvním funkčním blokem je převod do barevného prostoru $\mathrm{YC}_B\mathrm{C}_R$ podle rovnic (2.1). Dále v pořadí je volba použité vlnky následovaná blokem použité transformace. Po provedení transformace je potřeba dané koeficienty kvantizovat z reálných do celých čísel. Takto kvantizované koeficienty jsou následně zpracovány komprimačním algoritmem. Výsledný bitový tok je poté ještě možno zpracovat aritmetickým kodérem. Takto komprimovaný bitový tok je spolu s údaji pro inverzní funkce uložen do souboru. Součástí knihovny je i dekomprese obrazu a jedná se o identický postup,který je samozřejmě v inverzním pořadí.

Vstupy a výstupy

Samotná knihovna slouží pouze poskytnutí funkcí pro komprimaci a dekomprimaci obrazu. Komunikaci s knihovnou bude tedy realizovat demonstrační aplikace. Tato aplikace bude



Obrázek 4.1: Blokové schéma architektury knihovny.

brát parametry od uživatele a volat příslušné funkce knihovny.

Mezi parametry patří soubor na disku obsahující obraz, zvolené algoritmy pro transformaci a kompresi, kvalitu požadované komprese a název souboru pro uložení výstupu. Je také nutné specifikovat jestli se bude provádět komprese či dekomprese.

Jako poslední funkce demonstrační aplikace bude i výpočet kvality dekomprimovaného obrazu podle metod uvedených v kapitole 2.3. Výsledky těchto metod budou zobrazeny na standardním výstupu.

4.1.1 Platforma, prostředí a nástroje

Implementace projektu začala ve *Windows* pomocí prostředí *Cygwin*. Posléze pro zjednodušení různých závislostí byl překlad a testování přesunuto do operačního systému *Linux* pomocí aplikace *VirtualBox* verze 4.2.18, ve kterém již setrvalo. Konkrétně se jednalo o operační systém Ubuntu 14.04 LTS. Tento přesun umožnil snadnější využití různých utilit, které nebyly dostupné nebo špatně použitelné v předešlém prostředí.

Jako textový editor pro psaní zdrojových kódu bylo použito prostředí *Code::Blocks* verze 12.11, psaní skriptů shellu, vlastního obsahu této technické zprávy a další případné úpravy formátu souborů byly uskutečněny pomocí programu *Notepad++* verze 6.7.5. Všechny grafy byly vygenerovány pomocí utility *gnuplot* verze 4.6.

Tvorba různých obrázků a grafů (vyjma těch vygenerovaných aplikací) v této technické zprávě byla provedena v prostředí *Windows* pomocí programu *Inkscape* verze 0.48.4 r9939. Samotný překlad aplikace a knihovny byl uskutečněn pomocí překladačů *gcc* a *g++* verze 4.8.2. Sestavení knihovny bylo provedeno pomocí utility *ar* verze 2.24.

4.2 Aplikace

Účelem aplikace je obstarat rozhraní pro knihovnu obsahující metody pro zakódování a dekódování obrazu. Dále aplikace umožňuje porovnat dva obrazy pomocí metody SSIM (viz kapitola 2.3).

Aplikace (soubor *main.cpp*) očekává několik vstupních parametrů. Parametry i, o a f slouží k určení jmen souborů pro vstupní, výstupního a zakódovaný obraz. Kvalita komprese je určená parametry q a b, jenž jsou následovanými hodnotami v rozsahu specifickými na dané metodě. První parametr q určuje použití tvrdého prahování, kde je ponecháno pouze dané procento největších koeficientů. Očekávaná hodnota je tedy v rozsahu 0 až 100. Druhý parametr udává kolik bitových rovin se bude kódovat pomocí algoritmu pro kódování koeficientů. Jelikož se pracuje s hodnotami o velikosti 32 bitů, tak je očekávaná hodnota v rozsahu 0 až 31. Typ transformace je určen parametry w, c a s, které odpovídají

vlnkové, konturletové a shearletové transformaci. Zvolená vlnka pro vlnkovou transformaci je CDF 9/7, která je často využívána právě pro ztrátovou kompresi. Parametr e umožňuje použití algoritmu EZW pro zpracování koeficientů. Kvantizační koeficient je možné určit parametrem k v rozsahu $\langle 0, 1\rangle$. Povinné parametry jsou pouze pro názvy souborů, tj. i, o a f. Pokud nejsou udány žádné další parametry, tak je použita vlnková transformace, kvalita prahování 100 %, hodnota bitové roviny 31 a pro kódování koeficientů algoritmus SPIHT. Vyjímku tvoří situace při zadaném parametru m, při kterém proběhne pouze porovnání dvou obrazů metodou SSIM. Povinnými parametry jsou v tomto případě pouze i a o.

Výstupem aplikace jsou soubory zakódovaného a dekódovaného obrazu (jména určená vstupními parametry. Dále je na standardní výstup vypsána daná kvalita komprese a hodnota SSIM pro vstupní a dekódovaný obraz. Kvalita komprese je podle zvolené metody, přičemž v případě zadaných obou hodnot je zobrazena hodnota bitové roviny.

Jako externí závislost aplikace je pouze knihovna *OpenCV* [2], pomocí které je manipulováno s obrazy a vypočítána hodnota SSIM.

4.3 Knihovna

Knihovna pro kódování a dekódování vstupních souborů je implementovaná v souboru *mainLib.cpp*. Soubor obsahuje dvě hlavní funkce pro zakódování `encode_func` a dekódování `decode_func` společně s několika podpůrnými funkcemi. V těchto hlavních funkcích jsou prováděny operace podle blokového schématu na obrázku 4.1, v případě dekódování se jedná o inverzní postup.Podpůrné funkce představují mezistupeň mezi voláním příslušné funkce v dalších souborech. Jejich hlavním účelem je zpřehlednění kódu hlavních funkcí a oddělení příkazu specifických pro danou větev ve vykonávacím řetězci.

Práce se vstupním obrazem, jako je jeho načtení, převod a rozdělení barevných složek, je uskutečněna pomocí knihovny *OpenCV*. Po vykonání transformace, případně do vykonání inverzní transformace, jsou již koeficienty předávány pomocí datového typu *vector*. Pro odlišení jednotlivých cest v blokovém schématu architektury, jsou v hlavičkovém souboru *mainLib.h* definovány konstanty pro transformace `WAVELET`, `CONTOURLET` a `SHEARLET`. Stejně jsou definovány konstanty `EZW` a `SPIHT` pro algoritmus komprese koeficientů.

Mezi podpůrné funkce patří například volání vlnkové transformace v externí knihovně `transformDWT`. Tato funkce převede vstupní matici z formátu `cv::Mat` knihovny *OpenCv* do dvourozměrného vektoru `vector<vector<double> >` ze standardní knihovny *¡vector¿* a dále spočítá hloubku rozkladu na základě velikosti vstupní matice. Podpůrná funkce pro rekonstrukci pomocí vlnkové transformace `reconstructDWT` zase naopak převádí výsledný obraz z dvourozměrného vektoru na matici formátu `cv::Mat`.

Podobně jsou zde přítomny funkce pro konturletovou transformaci. Dopředná transformace je zavolána pomocí funkce `transformCT`, která převádí vstupní obraz z formátu `cv::Mat` knihovny *OpenCV* do formátu `mat` knihovny *libit*. Opačnou konverzi a volání inverzní transformace zajišťuje funkce `reconstructCT`. Výstupem, resp. vstupem, funkce konturletové transformace (funkce volaná funkcí `transformCT`), resp. inverzní konturletové transformace, je speciální datová struktura `contourlet_t`. Uniformní typ dat mezi jednotlivými bloky (obrázek 4.1) zajišťují převodní funkce `contourletToVector`, která převede pole výstupních matic do vektoru a k němu vytvoří nový vektor délek podobného formátu jako u vlnkové transformace (popsaný dále). Inverzní postup je implementován ve funkci `vectorToContourlet`.

Shearletová transformace nevyžaduje žádnou mezifunkci pro volání dopředné či zpětné transformace, ale stejně jako konturlety potřebuje převod na stejný výstupní, resp. vstupní,

typ dat. Tento převod mezi formáty zajišťuje dopředná funkce `shearletToVector` a k ní zpětná funkce `vectorToShearlet`.

Koeficienty transformace jsou kvantizovány z reálných čísel (datový typ `double`) na celá čísla s přesností 32 bitů (datový typ `int32_t`). Funkce kvantizace se nazývá `quantization` a jsou z ní volány další podpůrné funkce `maxDouble` a `minDouble` pro zjištění největší absolutní hodnoty v koeficientech. Zde je také uplatněno tvrdé prahování, kde koeficienty nižší než daný práh jsou vynulovány. Samotná kvantizace je poté provedena na všech prvcích podle rovnice (4.1), kde `MAX` udává největší absolutní hodnotu koeficientů a `MAX_INT32_T` reprezentuje největší kladnou hodnotu datového typu `int32_t`.

$$q_{i,j} = \text{round}\left(\left(\frac{c_{i,j}}{\text{MAX}}(2^{31}-1)\right)\cdot k\right) \tag{4.1}$$

Inverzní postup kvantizace je proveden funkcí `dequantization`, jenž pouze provede zpětný převod podle rovnice (4.2).

$$c_{i,j} = \left(\left(\frac{q_{i,j}}{2^{31}-1}\text{MAX}\right) \div k\right) \tag{4.2}$$

Prvním algoritmem pro kódování koeficientů je EZW. Každá transformace volá svoji specifickou dvojici funkcí. Pokud byla použita vlnková transformace, tak jsou volány funkce `ezwEncWT` a `ezwDecWT`. První funkce před voláním vlastní funkce pro kódování EZW zjistí práh největších koeficientů (zde se nezapočítávají aproximační koeficienty). Druhá funkce pro dekódování vypočítává a inicializuje délku výstupní posloupnosti koeficientů. Podobně pro konturletovou transformaci jsou zde přítomny funkce `ezwEncCT` a `ezwDecCT` a pro shearletovou transformaci funkce `ezwEncST` a `ezwDecST` se stejnou funkčností jako jejich vlnkové varianty.

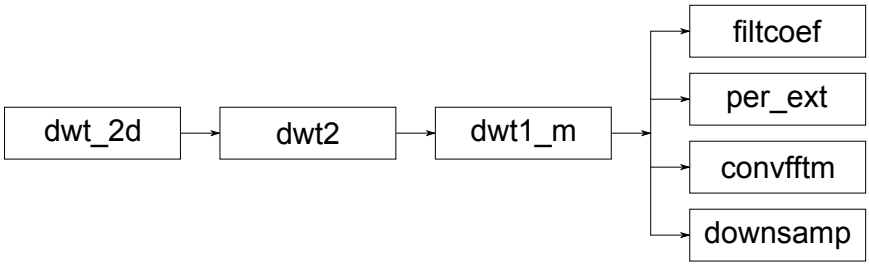
Druhým algoritmem pro kódování koeficientů je SPIHT. Všechny transformace zde již volají stejné funkce `spihtEnc` a `spihtDec`, jsou však mezi sebou odlišeny hodnotou parametru. Funkcionálně jsou stejné jako jejich EZW varianty, tj. první zjistí největší práh neaproximačních koeficientů a druhá inicializuje délku výstupní proměnné.

Externími závislostmi jsou knihovny *OpenCV* [2], *fftw3* [12] a *libit* [17].

4.3.1 Diskrétní vlnková transformace

Dopředná vlnková transformace je uskutečněna voláním mezifunkce jménem `transformDWT` a zpětná transformace naopak „mezifunkcí“ `reconstructDWT`, které jsou popsány výše. Vlastní implementace dopředné a zpětné vlnkové transformace se nachází ve složce `wavelet` v souboru s názvem `wavelet2s.cpp`. Tato implementace byla převzatá z [16].

Dopředná vlnková transformace je vykonána voláním funkce `dwt_2d`. Tato funkce provede dvourozměrnou separabilní vlnkovou transformaci konvolučním principem s periodickým rozšířením, tzn. výstupní „obraz“ bude mít stejné rozměry jako vstupní. Mezi parametry funkce patří kromě vstupního obrazu, počet dekompozičních úrovní, jméno použité vlnky ještě proměnná `flag` uchovávající hodnoty pro rekonstrukci inverzní funkcí. Jako posledními parametry je proměnná typu jednorozměrného vektoru pro výstup a délky. Tento vektor má daný formát $A(j)D_h(j)D_v(j)D_d(j)\cdots D_h(1)D_v(1)D_d(1)$, kde $A(j)$ udává aproximační matici a $D(k)$, kde $0 < k \leq j$, udává matici detailů na úrovni j. Matice detailů jsou vždy horizontální, vertikální a diagonální. Délky jednotlivých matic jsou uchovány v jednorozměrném vektoru.



Obrázek 4.2: Hierarchie volání funkcí dopředné vlnkové transformace.

Transformace probíhá po jednotlivých krocích, kde aktuální aproximační obraz je po každém kroku aktualizován (v prvním kroku je aproximačním obrazem vstupní obraz). Jedna úroveň vlnkové transformace je uskutečněna voláním funkce `dwt2`. Tato funkce reprezentuje separabilitu transformace pomocí jednorozměrné vlnkové transformace (funkce `dwt1_m`). Separabilita spočívá v aplikaci jednorozměrné transformace na každý řádek. Výsledkem jsou vektory pro nízké a vysoké frekvence. Na tyto výsledné vektory jsou pak znovu aplikovány jednorozměrné transformace, ale v tomto případě v dimenzi sloupců. Výsledkem jsou opět pro každý vektor dva vektory. Princip separabilní transformace je zobrazen na obrázku 3.6.

Jednorozměrná vlnková transformace implementována ve funkci `dwt1_m` provádí periodické rozšíření pomocí funkce `per_ext`. Následně je funkcí `convfftm` vypočítána konvoluce vstupního signálu s nízkopásmovým filtrem (získaným funkcí `filtcoef`). Pro výpočet Fourierovy transformace uvnitř funkce `convfftm` je použita externí knihovna *fftw3*. Výsledný signál z konvoluce je poté podvzorkován funkcí `downsamp`. Stejným postupem je vypočítána výstupní hodnota vysokofrekvenčního signálu. Hierarchie volání jednotlivých funkcí je zobrazena na obrázku 4.2.

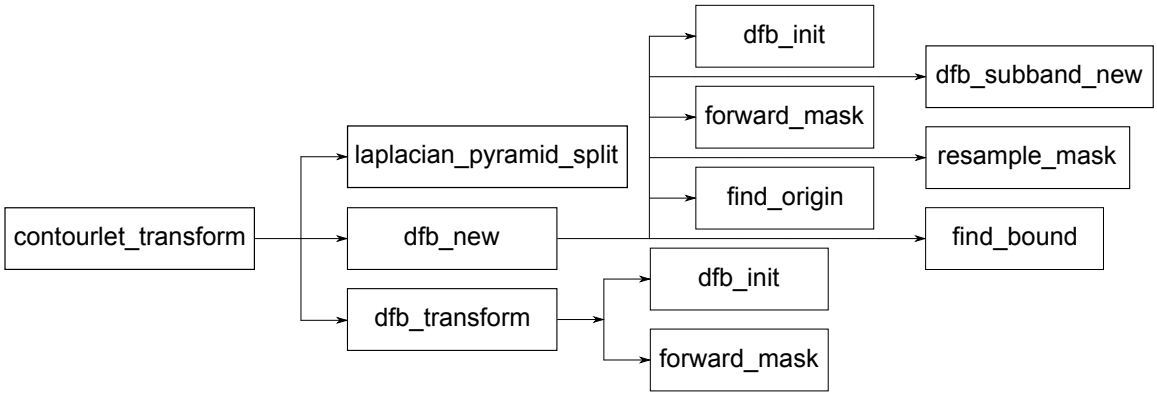
Zpětná transformace je dosažena funkcí `idwt_2d`, která požaduje jako parametry koeficienty z dopředné transformace ve formátu popsaném výše, název vlnky, proměnnou `flag` a délkový vektor vygenerovaný dopřednou transformací.

4.3.2 Kontourletová transformace

Pro implementaci konturletové transformace bylo vycházeno z implementace k článku [7] v jazyce C. Implementace se nachází ve složce *contourlet*. Hlavní odlišností je nepoužití vlnkové transformace pro aproximační matici a rozklad konturletové transformace vždy na co největší možnou úroveň.

Před samotnou transformací je nutno inicializovat datovou strukturu `contourlet_t` (definovanou v souboru `contourlet.h`) funkcí `contourlet_new`. Funkce provede alokaci paměti pro datovou strukturu a všechny rozkladové podobrazy. Dále inicializuje hodnoty pro úrovně rozkladu LP, SBF a hodnoty pyramidových filtrů. Pro konečné uvolnění paměti je k dispozici funkce `contourlet_delete`.

Konturletová transformace je implementována ve funkci `contourlet_transform`. Jako parametry přijímá vstupní matici obrazu v datovém typu z knihovny *libit* a inicializovanou datovou strukturu `contourlet_t`. Každý stupeň rozkladu se skládá z několika kroků. V prvním kroku je aktuální aproximační obraz rozložen Laplaceovou pyramidou na nízkofrekvenční a vysokofrekvenční obraz. Nízkofrekvenční obraz se uchová pro další stupeň rozkladu (v případě posledního stupně se na konci uloží do výstupní datové struktury). Vysokofrekvenční obraz je podroben analýze směrovou bankou filtrů vytvořenou funkcí `dfb_new`. Samotná směrová dekompozice je provedena funkcí `dfb_transform`. Dvě úrovně



Obrázek 4.3: Hierarchie volání funkcí dopředné konturletové transformace.

rozkladu jsou zobrazeny na obrázku 3.13. Zpětná transformace s inverzním postupem je implementována ve funkci `contourlet_itransform`.

Funkce pro inicializaci, destrukci, dopřednou a zpětnou transformaci jsou implementovány v souboru `contourlet.c`.

Laplaceova pyramida

Implementace LP je složena ze dvou funkcí. První funkce `laplacian_pyramid_split` reprezentuje pyramidový rozklad vstupního obrazu na nízkofrekvenční obraz poloviční rozměrů a vysokofrekvenční obraz stejného rozměru jako vstupní obraz. Naopak funkce `laplacian_pyramid_merge` provádí rekonstrukci původního obrazu z nízko a vysokofrekvenčního obrazu.

Směrová banka filtrů

Pro použití směrové banky filtrů je nutno vždy danou banku filtrů vytvořit funkcí `dfb_new`, a poté ji lze použít ve funkci analýzy `dfb_transform` nebo syntézy `dfb_itransform`. Samotná aplikace filtrů je realizována makrem `FILTER`.

Konstrukce směrové banky (pro každou úroveň rozkladu LP zvlášť) je realizována funkcí `dfb_new`. Parametry funkce jsou pouze rozměry aktuálního vysokofrekvenčního obrazu a výška binárního stromu, kde počet jeho listů udává počet směrů (například pro strom výšky 2 jsou 4 směry). Výstupní struktura `dfb_t` je definována v hlavičkovém souboru `dfb.h`. Funkcí `dfb_init` je nastavena symetrie filtrů. Po inicializaci rozměrů a výšky stromu je vytvořena dočasná maska funkcí `bmat_new`. Také je alokována paměť pro binární strom podpásem typu `dfb_subband_t` (definice struktury v souboru `dfb.h`) a počátky podpásem.

Adresování směrových podpásem je realizováno makrem `dfb_band`. Alokace jednotlivých podpásem je provedena voláním funkce `dfb_subband_new`. Inicializace matice koeficientů v jednotlivých podpásmech (člen struktury s názvem `image`) je provedena funkcí `mat_new` z knihovny *libit*. Směrová dekompozice je prováděna v cyklu jako průchod celým binárním stromem směrové dekompozice.

Před samotným cyklem je nejprve inicializována maska (člen struktury `mask`) kořenu stromu. Každý cyklus se skládá z několika kroků. Prvním krokem je inicializace vzorkovací a rotační matice a alokace potomků. V případě konečné úrovně je ještě provedena korekce

těchto dvou matic. Druhým krokem je výpočet nové masky funkcí `forward_mask` pro oba potomky. Opět je zde korekce pro poslední úroveň funkcí `resample_mask` a nalezení počátku a hranic v dané masce funkcemi `find_origin` a `find_bound`.

Transformace SBF je implementována ve funkci `dfb_transform` a skládá se ze dvou částí. Inicializace spočívá v nahrání vstupních koeficientů do kořene binárního dekompozičního směrového stromu. V prvním kroku jsou vypočteny jednotlivé dekompozice uvnitř stromu pomocí funkcí `direct_poly_filter_downsample` a korekce na poslední úrovni funkcí `forward_resample`. V druhém kroku ze stromu vyextrahovány jednotlivé směrové dekompozice z poslední úrovně a uloženy do výstupního pole matic.

Obdobně je implementována inverzní funkce `dfb_itransform`, která v prvním kroku uloží směrové dekompozice do listů dekompozičního směrového stromu. V druhém kroku zrekonstruuje dané matice koeficientů v uzlech stromu pomocí funkcí `backward_resample` a `direct_poly_interpolate_upsample`. Posledním krokem je pouhá extrakce původní matice z kořene stromu.

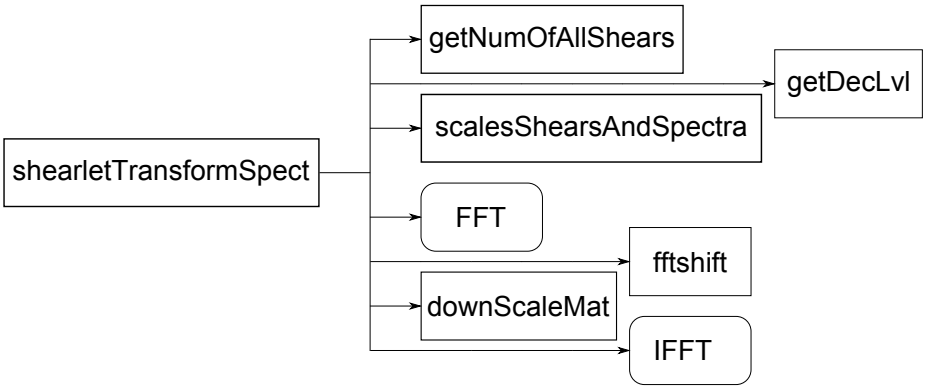
4.3.3 Shearletová transformace

Implementace shearletové transformace byla vytvořena na základě implementace *FFST - Fast Finite Shearlet Transform* k článku [14]. Vlastní implementace se nachází ve složce `shearlet`. Hlavní rozdíl spočívá v přidání podvzorkování a úprava počtu a tvorby shearletů. Pro vstupní a výstupní data je použit datový typ `cv::Mat` z knihovny *OpenCV*.

Jako u jediné implementované transformace zde není přítomna mezifunkce. Pouze je třeba udat velikost pole pro výstupní matice. Počet výstupních matic (ne velikost) je získán funkcí `getNumOfAllShears`. Samotná transformace je implementována ve funkci `shearletTransformSpect`, podobně zpětná transformace s opačným průběhem je implementována ve funkci `inverseShearletTransformSpect`.

Dopředná shearletová transformace se skládá z několika částí. Jako první se vypočítá maximální možná úroveň rozkladu daného rozměru obrazu funkcí `getDecLvl` a počet shearletů pro danou velikost funkcí `getNumOfAllShears`. Dalším krokem je tvorba shearletů pomocí funkce `scalesShearsAndSpectra`, která je podrobněji popsána níže. Ještě před transformací jsou hodnoty vstupního obrazu převedeny na datový typ `double` (konkrétně `CV_64FC1` z knihovny *OpenCV*). Samotná transformace začíná aplikací dvourozměrné dopředné Fourierovy transformace na vstupní obraz. Pro Fourierovu transformaci je použita knihovna *fftw3*. Konkrétně je použit plán (funkce) `fftw_plan_r2r_2d`, jenž má vstupní i výstupní data typu reálných čísel (nikoli komplexní čísla). Výsledné koeficienty jsou v kartézských souřadnicích, ale shearlety jsou v pseudo–polární. Převod do pseudo–polární mřížky a zpět do kartézské provádí funkce `fftshift`. Koeficienty jsou dále násobeny s maticemi všech shearletů. Zde shearlety fungují jako váhová maska. Jednotlivé výsledné matice jsou poté volány ve funkci `downScaleMat`, která odstraní nulové okraje (netýká se shearletů na nejvyšší úrovni). Posledním krokem je zpětný převod z frekvenčního spektra. Převod je proveden zpětnou Fourierovou transformací opět z knihovny *fftw3* na všechny vzniklé podobrazy. Nutno upozornit, že se nejedná o ryze inverzní transformaci k předešlé (jiné rozměry matic), a proto zde není normalizace. Výsledné matice jsou poté uloženy do výstupního pole.

Zpětná shearletová transformace provádí podobný postup jako dopředná. Úroveň rozkladu, shearlety a jejich počet je vypočítán stejně jako v opačném směru. Na každý vstupní podobraz je aplikována dopředná Fourierova transformace z knihovny *fftw3*. Jelikož se jedná o inverzní operaci k druhé Fourierově transformaci z dopředného směru, je nutné prvky vý-



Obrázek 4.4: Volané funkce a bloky Fourierovy transformace dopředné shearletové transformace.

sledné matice normalizovat hodnotou $4 * m * n$, kde m, n jsou rozměry dané matice. Po normalizaci nastává opět převod do pseudo–polárních souřadnicové systému funkcí `fftshift`. Jednotlivé matice jsou poté poslány do funkce `upScaleMat`, která jim navrátí původní velikost (velikost původního obrazu) přidáním nulových okrajů a hned jsou matice v původní velikosti násobeny s příslušnými shearletami. V dalším kroku jsou všechny matice sečteny do nové matice, na kterou je použita zpětná Fourierova transformace. Jelikož se jedná o inverzní operaci k první Fourierově transformaci z funkce `shearletTransformSpect`, tak je nutno jednotlivé výsledné hodnoty normalizovat hodnotou $4 * m * n$, kde m, n jsou rozměry dané matice. Výsledná matice je převedena na typ `CV_8UC1` (z knihovny *OpenCV*) a uložena do výstupní matice.

Převod funkcí `fftshift` do pseudo–polárních souřadnic je proveden křížovým prohozením levého horního kvadrantu s pravým spodním a pravého horního kvadrantu s levým spodním.

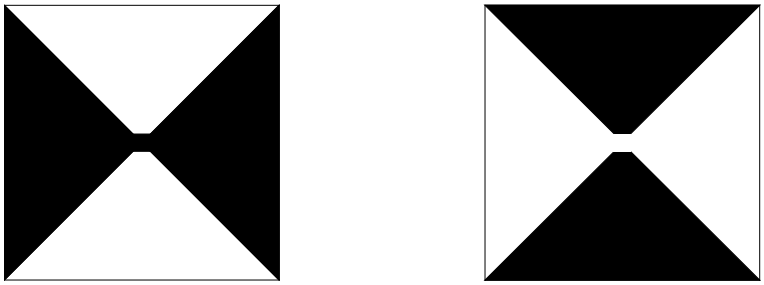
Odstranění nulového okraje funkcí `downScaleMat` je proveden na základě průchodu shearlety všemi řádky a sloupci z obou stran. Řádky nebo sloupce obsahující pouze nuly jsou vynechány. V průchodech se spočte počet nul od začátku průchodu. Ze všech průchodů řádků se poté vybírá nejmenší hodnota, tzn. vybere se nejširší řádek. Stejně se vybere nejvyšší sloupec. Z těchto dvou hodnot (nejširší řádek a nejvyšší sloupec) se vybere menší hodnota, která určuje šířku nulového okraje v dané shearletě. Následně se odstraní daný nulový okraj metodami `rowRange` a `colRange` knihovny *OpenCV* v matici koeficientů.

Funkce `upScaleMat` obstarává obnovu původní velikosti dané matice. Velikost okraje je spočtena na základě velikosti aktuální matice a známé původní velikosti. Přidání nulového okraje je realizováno funkcí `cv::copyMakeBorder` z knihovny *OpenCV*.

Shearlety

Tvorba shearlet představuje hlavní jádro shearleové transformace. Základní princip generování každé shearlety se skládá ze dvou kroků. Prvním krokem je vymezení prostoru v matici (pracuje se v pseudo–polárních souřadnicích) a druhým krokem je aplikace vlnky na tento vymezený prostor. V implementaci je použita Meyerova vlnka. Při tvorbě shearletů se nepracuje se vstupním obrazem, ale pouze jeho rozměry. Pro práci s maticemi je použit datový typ a funkce z knihovny *OpenCV*.

K určení polohy je použita metoda vytvoření dvou „souřadnicových“ matic (pro osu x a y) velikosti vstupního obrazu funkcí `getMeshgrid`, která naplní matice hodnotami lineárně



Obrázek 4.5: Horizontální a vertikální matice kuželových výsečí.

rozloženými po dané ose s nulou uprostřed (nuly jsou ve skutečnosti dvě, kvůli lepší symetrii u sudých rozměrů). V dalším kroku jsou na základně souřadnicových matic vytvořeny nové matice kuželovitý výsečí, jak je znázorněno na obrázku 4.5. Výseče jsou v matici reprezentovány jedničkou a zbylé hodnoty jsou nulové. V posledním kroku inicializace je získán vektor počtu shearlet (mimo aproximační) a jejich celkový počet (včetně aproximační) funkcemi `getShearsPerScale` a `getNumOfAllShears`.

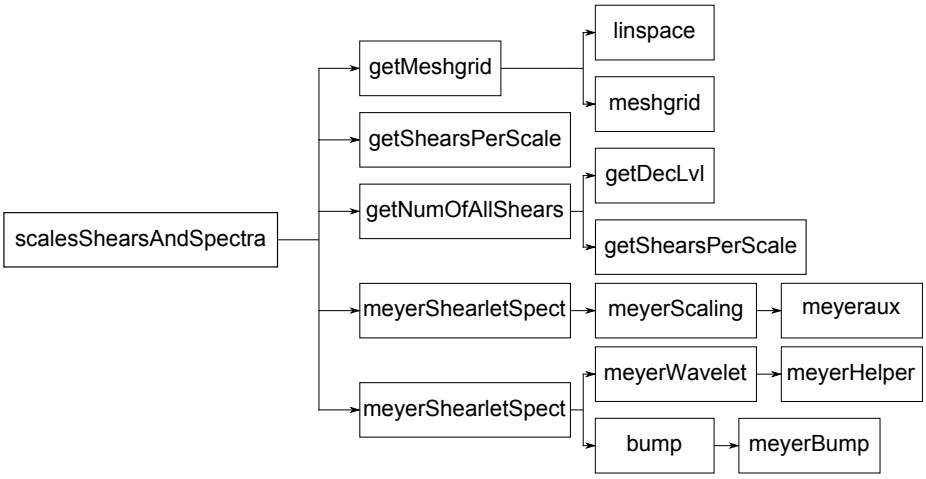
Na rozdíl od konturletové transformace je u shearletové transformace postupováno od nejmenšího aproximačního rozměru velikosti 2×2 až po největší původní rozměr vstupního obrazu. Aproximační shearleta je získána funkcí `meyerShearletSpect` s posledním parametrem nastaveným na hodnotu `true`. Tato funkce vytvoří aproximační shearletu na základě předaných souřadnicových matic vytvořených dříve.

Úrovně rozkladu (soustředné čtverce v obrázku 3.16) jsou poté tvořeny postupně v cyklu. Jednotlivé směry (a tedy shearlety) jsou generovány v cyklu různým násobením kuželových a vlnkových matic. Vlnkové matice jsou vytvořeny funkcí `meyerShearletSpect` s posledním parametrem hodnoty `false`. Dalšími parametry kromě souřadnicových matic jsou vzorkovací a směrový koeficient. Vzorkovací koeficient hodnoty 2^{-2*j}, kde j udává danou úroveň. Směrový koeficient nabývá hodnot od -1 do 1. Tyto hodnoty vygenerují na každé úrovni čtyři směry. Počet směrů je takto omezen pro názornější srovnání s konturletovou transformací. Počet směrových cyklů je pouze tři, neboť diagonální směry shearlet jsou vypočteny ve stejném cyklu.

Shearlety jsou na konci uloženy ve výstupním poli matic seřazené od aproximační po poslední na nejvyšší úrovni rozkladu. Na jednotlivých úrovních jsou shearlety seřazené od horizontální po vertikální v protisměru hodinových ručiček.

Souřadnicové matice

Souřadnicové matice jsou generovány na základě velikosti vstupního obrazu pomocí funkce `getMeshgrid`. Podle počtu dekompozičních úrovní je spočtena maximální hodnota a je vytvořen vektor od nuly do této maximální hodnoty s lineárním rozložením funkcí `linspace`. Prakticky pro rozměry mocnin dvou jsou kroky po 0.5. Výsledný vektor je následně zrcadlen do záporných čísel (včetně nuly) pro vytvoření číselné osy (vektor začíná záporným maximem). Takto byl vytvořen vektor pro osu x a vektor pro osu y je pouze převrácený (začíná kladným maximem). Souřadnicové matice jsou z vektorů vytvořeny pouhým opakováním funkcí `meshgrid`.



Obrázek 4.6: Hierarchie volaných funkcí při vytváření shearlet.

Aproximační a vlnkové matice

Aproximační i vlnkové matice jsou generovány stejnou funkcí `meyerShearletSpect` s rozdílem v posledním parametru. Dalšími parametry jsou souřadnicové matice, vzorkovací a směrový koeficient. U aproximační matice nejsou koeficienty zapotřebí a jsou pro ně použity hodnoty NAN. Naopak u vlnkových matic je vzorkovací koeficient roven hodnotě 2^{-2*j}, kde j udává danou úroveň a směrový koeficent nabývá hodnot $-1, 0, 1$.

Aproximační matice (shearlet) je vytvořena pomocí matic kuželových výsečí pro horizontální a vertikální směr `C_hor` a `C_hor` a vlnkové matice vytvořenou funkcí `meyerScaling` s parametrem souřadnicové matice příslušného směru (x nebo y). Funkce `meyerScaling` před samotným výpočtem vytvoří dvě matice (`int1` a `int2`) pro lokalizaci sousedních podprostorů v daném rozmezí vzhledem k aktuální úrovni rozkladu. Hodnoty v prvním podprostoru jsou dány lokalizační maticí `int1` a v druhém prostoru hodnotami z volání funkce `meyeraux` lokalizované maticí `int2`. Oba podprostory jsou následně sjednoceny (sečteny). Funkce `meyeraux` provádí výpočet pomocné funkce Meyerovy vlnky podle rovnice (4.3).

$$v(x) = 35x^4 - 84x^5 + 70x^6 - 20x^7 \tag{4.3}$$

Vlnková matice je vytvořena ve funkci `meyerShearletSpect` s posledním parametrem hodnoty `false`. V prvním kroku se vytvoří nové souřadnicové matice podle daného vzorkovacího a směrového koeficientu. Samotná tvorba vlnkové matice je provedena podle rovnice (4.4). Rovnice (4.5) zabraňuje dělení nulou.

$$P = \text{meyerWavelet}(aX) \times \text{bump}\left(\frac{\sqrt{a}(s\sqrt{a}X + \sqrt{a}Y)}{xx}\right) \tag{4.4}$$

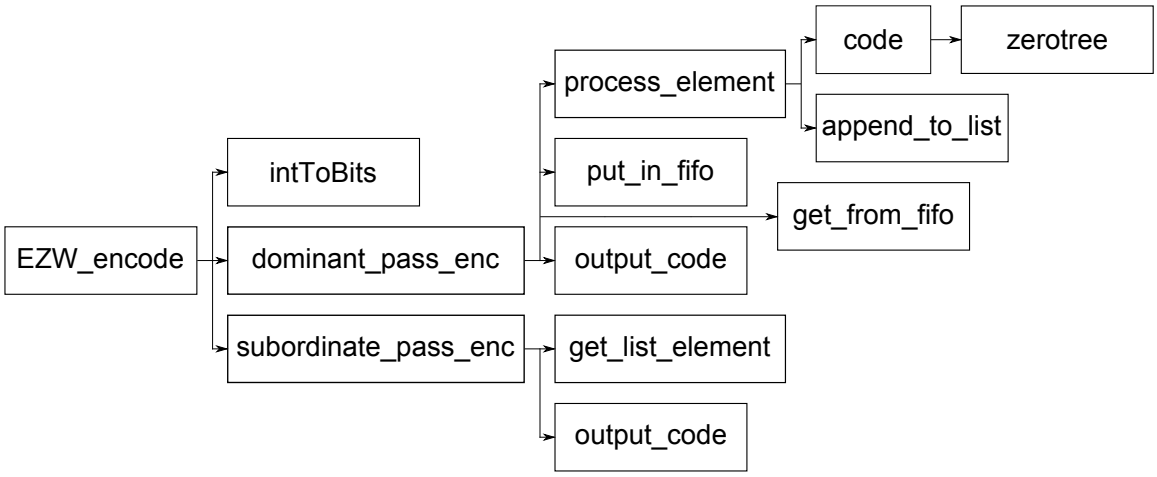
$$xx(i,j) = \begin{cases} aX(i,j) & \text{pro } aX(i,j) \neq 0, \\ 1 & \text{pro } aX(i,j) = 0 \end{cases} \tag{4.5}$$

Funkce `meyerWavelet` vypočte podprostor pomocí funkce `meyerHelper` ve vlnkové matici, kde budou platné hodnoty. Funkce `bump` vytváří s pomocí funkce `meyerBump` hodnoty vlnkové matice.

4.3.4 EZW

Pro implementaci algoritmu EZW byly jako základní kostra použity zdrojové kódy z webové stránky [33]. Hlavním rozdílem je odstranění omezení použití pouze pro vlnkovou transformaci a specifické části byly zobecněny a doplněny speciálními funkcemi pro odlišení použité transformace. Zdrojové kódy jsou k nalezení ve složce `ezw`.

Kód dopředného zakódování algoritmem EZW je implementován v souboru `ezwEnc.cpp`. Algoritmus využívá datové struktury pro seznam a frontu. Definice a implementace obslužných funkcí je v souborech `list.cpp` a `fifo.c` a příslušných hlavičkových souborech. Kódování začíná inicializací seznamu a fronty. Aproximační matice je zvlášť uložena do výstupního bitového toku funkcí `intToBits`. Vlastní algoritmus je poté proveden v cyklu pro postupně se snižující práh. Konečná hodnota prahu je určena vstupním parametrem bitové roviny. Cyklus algoritmu se skládá ze tří kroků, dominantní průchod, podřadný průchod a posunutí prahu na polovinu.

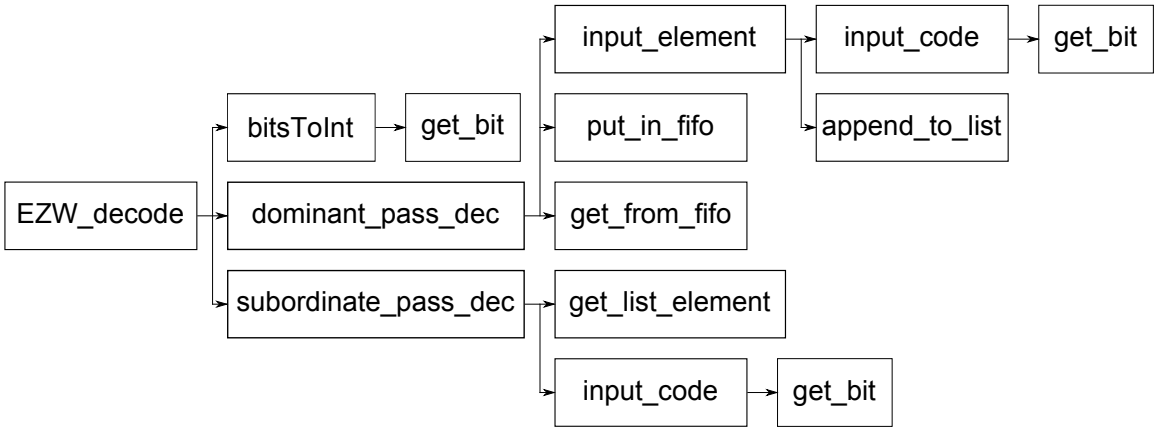


Obrázek 4.7: Hierarchie volaných funkcí při kódování EZW.

Dominantní průchod implementovaný ve funkcí `dominant_pass_enc` obsahuje dvě části. V inicializaci jsou zpracovány první matice úrovně rozkladu (nejhrubější detaily). Zpracováním je myšleno volání funkce `process_element` a její výsledek uložen do fronty funkcí `put_in_fifo`. V hlavní části dominantního průchodu je cyklus, který se opakuje pokud je fronta neprázdná. Pokud je fronta neprázdná, tak se vyjme první prvek funkcí `get_from_fifo` a pokud se nejedná o symbol pro nulový strom (symbol `ZTR`), tak jsou zpracováni jeho potomci (pokud nějaké má). Pokaždé když je z fronty vyjmut prvek, tak je jeho symbol zakódován do výstupního bitového toku funkcí `output_code`.

Zpracování koeficientu funkcí `process_element` znamená zjištění příslušného symbolu na základě hodnoty daného prvku a případně jeho potomků funkcí `code`. Pokud byl symbol pro daný prvek `POS` nebo `NEG`, pak je uložena jeho absolutní hodnota do seznamu funkcí `append_to_list` a ve vstupní posloupnosti je tento prvek vynulován.

Funkce `code` přiděluje prvku jeden ze symbolů `POS`, `NEG`, `IZ` a `ZTR`. První dva symboly jsou přiděleny, pokud je absolutní hodnota prvku větší než aktuální práh. Poslední dva sym-



Obrázek 4.8: Hierarchie volaných funkcí při dekódování EZW.

boly jsou přiděleny pokud nebyla předešlá podmínka splněna a na základě výsledku funkce `zerotree`, jenž zkoumá zda se jedná o nulový strom. To je docíleno ve funkci `zerotree` rekurzivním voláním na všechny potomky. Pokud byly všichni potomci byli nuloví, pak se jedná o nulový strom, jinak je aktuální prvek izolovanou nulou.

Podřadný průchod (funkce `subordinate_pass_enc`) je vždy prováděn s polovičním prahem než dominantní. Jedná se o průchod celým seznamem a každý prvek v seznamu získaný funkcí `get_list_element` je zakódován funkcí `output_code` bit určený daným prahem.

Zpětné dekódování algoritmem EZW je implementováno funkcí `EZW_decode` v souboru `ezwDec.cpp`. Tělo funkce je téměř totožné s dopředným směrem. Místo uložení aproximační matice je načtena z bitového toku funkcí `bitsToInt`. Cyklus dekódování je opět tvořen voláním dominantního a podřadného průchodu a posunem prahu. Získání dalšího bitu z bitového toku je implementováno funkcí `get_bit`.

Dominantní průchod dekódování je implementován ve funkci `dominant_pass_dec`. Opět jsou na začátku zpracovány do fronty matice první rozkladové úrovně, ale nyní je volána funkce `input_element`. Následný hlavní cyklus probíhá dokud není fronta prázdná. Vyjme se první prvek z fronty funkcí `get_from_fifo` a pokud se nejedná o nulový strom, tak se zpracují všechny jeho potomci.

Funkce `input_element` získá ezw symbol ze vstupního bitového toku funkcí `input_code` a pokud se jedná o symbol `POS`, tak do výstupní matice na danou pozici je uložena hodnota prahu, v případě symbolu `NEG` je uložena záporná hodnota prahu. V obou případech je pozice prvku uložena do seznamu funkcí `append_to_list`. Funkce `input_code` postupně čte bity z bitového toku funkcí `get_bit` vrací ezw symboly nebo hodnoty bitu podle toho, kolik bylo potřeba zpracovat bitů (v dominantním průchodu 2, v podřadném průchodu 1). Po získání prvku funkcí `input_element` je do fronty uložena struktura prvku a symbolu funkcí `put_in_fifo`.

Podřadný průchod ve funkci `subordinate_pass_dec` obsahuje cyklus pro všechny prvky seznamu. Pro každý prvek je volána funkce `input_code` a podle jejího výsledku je prvku na pozici, jenž je dána hodnotou aktuálního zpracovávaného prvku ze seznamu, přičten nebo odečten práh. Opět je funkce pro podřadný průchod volána s polovičním prahem než dominantní průchod ze stejného cyklu.

Aritmetický kodér

Výsledný bitový tok z EZW obsahuje stále určitou redundanci. Tato redundance lze ještě snížit aritmetickým kodérem. Základ implementace tvořily zdrojové kódy převzaté z webové stránky [20]. Implementace je k nalezení ve složce `ac` v souboru `ac.cpp` a příslušném hlavičkovém souboru. Kódování je prováděno voláním funkce `acEncode` a naopak dekódování funkcí `acDecode`.

Model pro začíná počítat se stejnou pravděpodobností výskytu jedniček a nul. Algoritmus kódování (funkce `acEncode`) probíhá v cyklu pro každý vstupní bit. V prvním kroku se nastaví hranice v aktuálním intervalu, tj. proměnné `a` a `b` a hranice podintervalů (proměnná `boundary`). Podle aktuálního vstupního bitu se upraví hranice nového intervalu. Pokud okrajové hranice překročí polovinu z maximálního možného intervalu, tak je na výstupní bitový tok poslán bit podle toho, z které strany se překročilo, a hranice aktuálního intervalu jsou upraveny. Po skončení cyklu pro všechny bity ve vstupním bitovém toku, nastává konečná fáze. V této fázi se podle toho, který podinterval je větší, posílají do výstupního bitového toku zakončovací bity.

Algoritmus pro dekódování je implementován ve funkci `acDecode`. Dekódování probíhá v cyklu pro celý vstupní bitový tok. V průběhu algoritmu jsou použity dva intervaly. První interval s hranicemi `u` a `v` a „středovým“ bodem `halfway` reprezentuje zakódovanou abecedu a druhý interval `a` a `b` a „středovým“ bodem `boundary` značí zdrojovou abecedu. V prvním kroku algoritmu je přečten bit a aktualizován interval zakódované abecedy. Dalším krokem je cyklus `do-while`, který probíhá dokud není potřeba aktualizovat aktualizovat interval zdrojové abecedy. První částí je aktualizace modelu zdrojové abecedy, konkrétně „středového“ bodu. V druhé části jsou aktualizovány hranice modelu zdrojové abecedy podle přečteného bitu ze vstupního bitového toku. Zároveň je nastaven příznak pro aktualizaci modelu. V poslední části vnitřního cyklu je další cyklu typu `while`, který simuluje práci kodéru při zpracovávání sledu jedniček nebo nul.

Kodér ani dekodér nevolají, žádné externí funkce. Kromě přístupových metod datového typu `vector` pro čtení (`at`) a zápis (`push_back`).

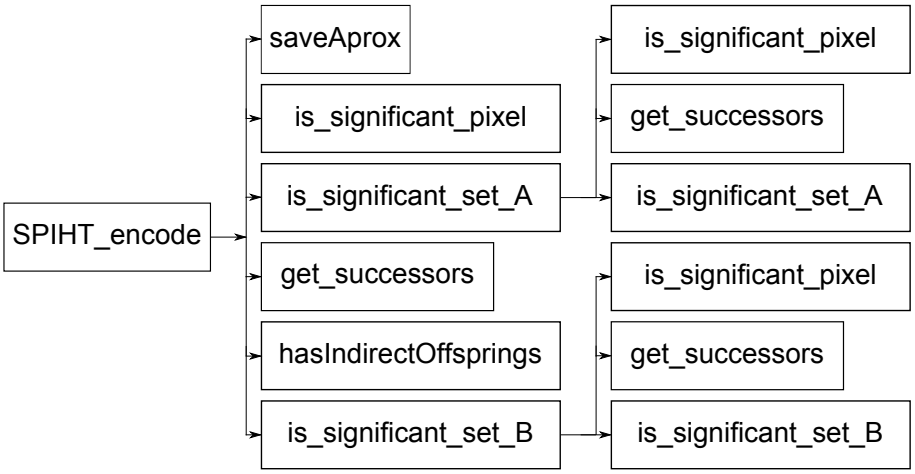
4.3.5 SPIHT

Základní kostra implementace byla převzata ze softwaru [24]. Hlavní změny spočívají v jiném zpracování inicializace a aproximační matice a zobecnění práce s potomky vlastními funkcemi pro dané transformace. Implementace se odráží od algoritmu 2.

V první části inicializace je nejdříve zpracována aproximační matice funkcí `saveAprox`. Následně jsou naplněny seznamy `LIP` a `LIS`. Odlišné zpracování aproximační matice způsobilo její vynechání při inicializaci seznamu `LIP`. Posledním krokem inicializace je určení konečného prahu, který mohl být ovlivněn vstupním parametrem pro hranici v bitových rovinách.

Řadící průchod začíná zpracováním prvků v seznamu `LIP` postupným voláním funkce `is_significant_pixel` a uložením jejího výsledku do výstupního bitového toku. Pokud je daný prvek významný, tak je do výstupu uloženo jeho znaménko a prvek je přesunut do seznamu `LSP`.

Po zpracování všech prvků seznamu `LIP` jsou zpracovány prvky seznamu `LIS`. Postup zpracování určuje typ prvku. U prvku typu A je volána funkce `is_significant_set_A` a pokud byl prvek významný, tak jsou zpracováni všichni jeho přímí potomci (získaní funkcí `get_successors`) funkcí `is_significant_pixel`. Všichni významní potomci jsou přesunuti do seznamu `LSP` a nevýznamní do seznamu `LIP`. Pro každého významného potomka je



Obrázek 4.9: Hierarchie volaných funkcí při kódování SPIHT.

uloženo do výstupu jeho znaménko. Na konec pokud má původní prvek významné nepřímé potomky, zjištěno funkcí `hasInderectOffsprings`, je přesunut do seznamu `LIS` jako prvek typu B. Daný prvek vždy ze seznamu `LIS` odebrán.

Pokud je aktuální prvek v seznamu `LIS` typu B, tak je zpracován, podobně jako u prvku typu A, funkcí `is_significant_set_B`. V případě, že prvek má významné nepřímé potomky, tak jsou tito potomci uloženi do seznamu `LIS` jako typ A. Aktuální prvek je ze seznamu `LIS` odstraněn.

Upřesňovací průchod zpracovává prvky seznamu `LSP` porovnáváním s prahem a případném uložení daného bitu prvku do výstupního bitového toku. Po skončení upřesňovacího průchodu před koncem cyklu je aktualizován práh.

Dekodér implementovaný ve funkci `SPIHT_decode` pracuje totožným způsobem jako kodér. Jediný rozdíl je kromě uložení bitu je bit načten ze vstupního toku. Jednotlivé seznamy jsou v každém kroku stejné jako u kodéru.

4.3.6 Potomci

Problémem kompatibility algoritmů pro kódování koeficientů a daných transformací je rozdíl ve struktuře výsledných koeficientů. Tedy hlavním problémem je přístup k potomků daného koeficientu, kde se u transformací může lišit jejich počet a pozice. Implementace funkcí identifikace potomků pro jednotlivé transformace jsou ve složce `descendants`. Jednotlivé funkce pro všechny transformace přijímají stejný počet parametrů (s vyjímkou vlnkové transformace).

Vlnková transformace má pevně daný počet matic detailů na dané úrovni rozkladu, a proto zde nemá smysl předávat tento parametr. Funkce `getDescendantsWT` v souboru `descendantsWT.cpp` získá indexy potomků pro daný vstupní index. Algoritmus získá číslo aktuální úrovně (funkce `getLevelWT`) a porovná ho s maximální možnou úrovní. Pokud daný index má potomky, tak je volána funkce `getLvlIndexWT` pro získání indexu začátku aktuální úrovně a pomocí známé délky řádku čísla sloupce v dané podmatici z funkce `getColNumWT` jsou vypočítány indexy čtyřech potomků.

V případě konturletové transformace je výpočet indexů potomků dále zobecněn. Implementace je k nalezení v souboru `descendantsWT.cpp` ve funkci `getDescendantsCT`. Výpočet indexů potomků se nepočítá z daného prvku, ale identifikuje se aktuální podmatice (get-

SubLevelCT) a index začátku dané podmatice v další úrovni (getNextLvlSubIndexCT). Řádek a sloupec daného prvku je získán funkcemi `getRowNumCT` a `getRowLenCT`. Zbylý postup pro výpočet indexu potomků je podobný jako u vlnkové transformace.

Koeficienty z shearletové transformace mají stejnou strukturu jako z konturletové. Postup identifikace indexů potomků je tedy téměř stejný s rozdílem v počtu potomků. Konturletová transformace podvzorkovává dvěma, z čehož plynou čtyři potomci. Shearletová transformace podvzorkovává čtyřmi, takže prvky mají 16 potomků. Implementace se nachází v souboru `descendantsST.cpp` ve funkci `getDescendantsST`.

4.3.7 Uložení a načtení souboru

Uložení a načtení výsledného bitového toku je implementováno v souboru `fileSL.cpp` ve složce `fileSL`. Uložení výstupního bitového toku je obstaráno funkcí `fileSave`. Před samotným uložením výsledného toku jsou uloženy metadata pro zpětnou rekonstrukci. Uložení bitového toku vykonává funkce `writeBits`. Výsledná struktura souboru je popsána v tabulce 4.1. Podobně načtení metadat a bitového toku je implementováno ve funkci `fileLoad` a bitový tok je přímo načten funkcí `readBits`.

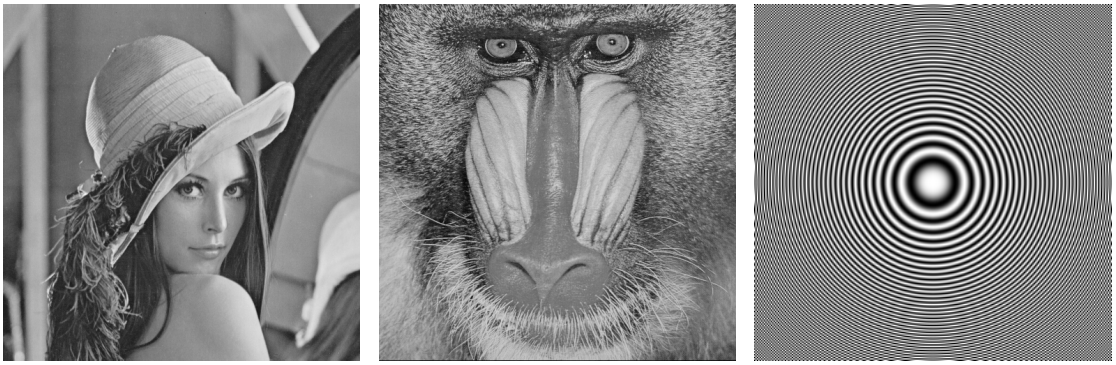
datový typ	hodnota	poznámka
int	typ transformace	
int	typ kódování	
int	počet bitových rovin	
double	kvantizační koeficient	
int+	délka a jméno vlnky	pouze vlnková trans.
2×int	výška a šířka matice Y	
2×int	výška a šířka matice C_B	
2×int	výška a šířka matice C_R	
3×double	maximum Y, C_B a C_R	
int+	délkový vektor Y	
int+	délkový vektor C_B	
int+	délkový vektor C_R	
int+	vektor příznaků Y	pouze vlnková trans.
int+	vektor příznaků C_B	pouze vlnková trans.
int+	vektor příznaků C_R	pouze vlnková trans.
int	počet úrovní rozkladu	pouze konturletová trans.
int	hloubka stromu směrového rozkladu	pouze konturletová trans.
3×int32_t	hodnota prahu jasové složky Y, C_B a C_R	
int	velikost bitového toku Y před AC	pouze EZW
int	velikost bitového toku C_B před AC	pouze EZW
int	velikost bitového toku C_R před AC	pouze EZW
vector::size_type	velikost bitového toku jasové složky Y	
vector::size_type	velikost bitového toku barevné složy C_B	
vector::size_type	velikost bitového toku barevné složy C_R	
	bitový tok jasové složky Y	
	bitový tok barevné složy C_B	
	bitový tok barevné složy C_R	

Tabulka 4.1: Struktura výstupního komprimovaného souboru

Kapitola 5

Srovnání

Tato kapitola obsahuje postup srovnávání jednotlivých algoritmů jak pro kódování koeficientů a určení kvality až po jednotlivé dříve probrané transformace. Postupným porovnáváním je dosaženo určení třídy obrazů, které jsou efektivnější zpracovat konturletovou nebo shearletovou transformací než vlnkovou.



(a) Lena (b) Baboon (c) Zoneplate

Obrázek 5.1: Počáteční množina testovacích obrazů.

Do počáteční množiny testovacích obrazů jsou zvoleny následující obrazy. Pro přirozený obraz fotografie je zvolen obrázek 5.1(a). Obraz 5.1(b) představuje obtížně komprimovatelný obrázek obsahující víceméně rovnoměrné zastoupení frekvencí od nízkých až po vysoké. Posledním členem počáteční testovací množiny je obraz 5.1(c), který obsahuje velké množství křivek.

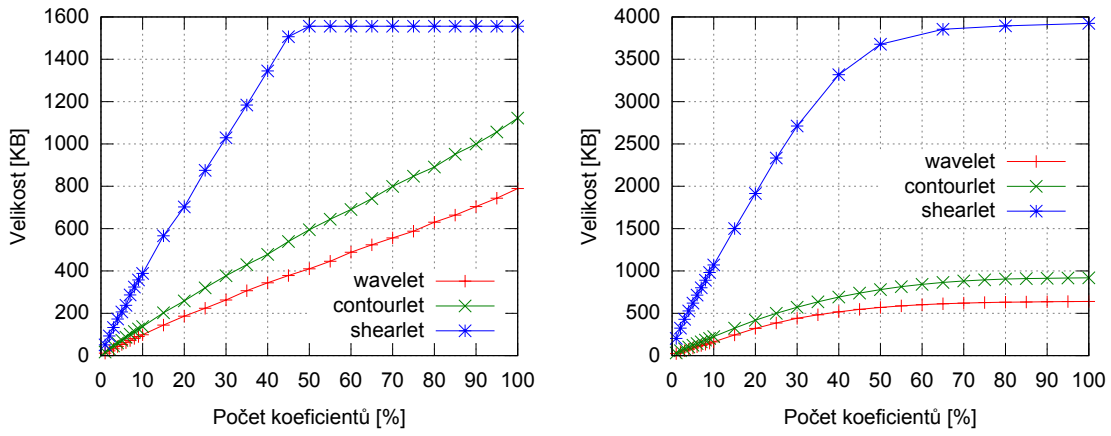
Spouštění, vstupní parametry

Demonstrační aplikace je postupně spouštěna s různými parametry pro získání výsledků v podobě souboru zakódovaného vstupního obrazu a hodnoty SSIM mezi vstupním a dekódovaným obrazem. Mezi povinné parametry patří cesta k vstupnímu obrazu (–i), cesta k uložení zakódovaného obrazu (–f) a cesta k uložené dekódovaného obrazu (–o). Při neuvedení žádných dalších parametrů bude provedena vlnková transformace s algoritmem pro kódování koeficientů SPIHT. Volba transformace probíhá pomocí třech parametrů, –w pro vlnkovou transformaci, –c pro konturletovou transformaci a –s pro shearletovou transformaci. Výchozí algoritmus pro kódování koeficientů je SPIHT, při zadání parametru –e je

nahrazen algoritmem EZW. Kvalita komprese je určena buď parametrem –b s hodnotou prahu pro kódování bitových rovin, nebo –q s hodnotou prahu pro nulování koeficientů při kvantizaci. Posledním parametrem je možnost zvolení kvantizačního koeficientu pomocí parametru –k o hodnotách z intervalu $\langle 0, 1\rangle$. V případě neuvedení parametru –k je kvantizační koeficient roven jedné.

Struktura koeficientů

Jednotlivé transformace se liší nejenom hodnotami výstupních koeficientů, ale také jejich strukturou. Koeficienty vlnkové transformace lze uspořádat podle principu na obrázku 3.7. Z toho plyne, že počet koeficientů je vždy stejný, jako počet pixelů v původním obraze. Jiná situace nastává u konturletové transformace. Směrová banka filtrů sice nenavyšuje počty koeficientů, ale Laplaceova pyramida navyšuje jejich počet až o 33 %. U shearletové transformace je navýšení koeficientů ještě větší. Při aplikaci shearlet na dané úrovni se vytváří několik podobrazů stejné velikosti. Při čtyřech shearletách na úrovni se počet koeficientů zvětší na více než čtyřnásobek. Většina koeficientů v podobrazech je ale rovna nule. V grafu 5.2 lze vidět rozdíly v počtu výstupních koeficientů pomocí tvrdého prahování, které například u shearletové transformace začíná účinkovat až od poloviny koeficientů.



(a) Kódování EZW

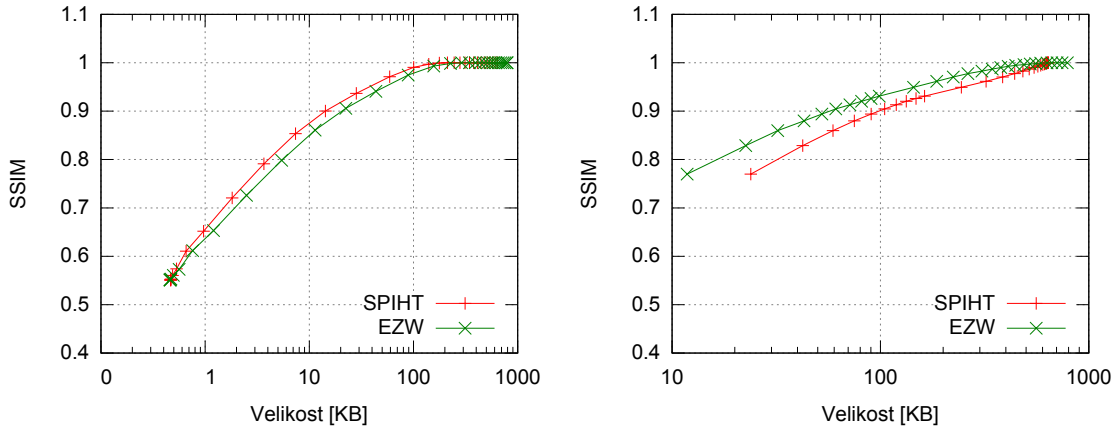
(b) Kódování SPIHT

Obrázek 5.2: Efekt tvrdého prahování na transformace na obraze Lena.

Kódování koeficientů

V knihovně jsou implementovány dva algoritmy pro kódování a dekódování koeficientů. Algoritmus EZW je myšlenkovým otcem algoritmu SPIHT, takže lze očekávat s poměrně dobrou pravděpodobností, že algoritmus SPIHT bude o něco efektivnější, než algoritmus EZW. Porovnání těchto dvou algoritmů je znázorněno v grafech 5.3.

Při určení kvality bitovou rovinou, jak je znázorněno v grafu 5.3(a) je algoritmus SPIHT podle očekávání efektivnější. Překvapivě při použití určení kvality tvrdým prahováním koeficientů, vyplývá z grafu 5.3(b) jako poměrně lepší varianta algoritmus EZW s aritmetickým kodérem.



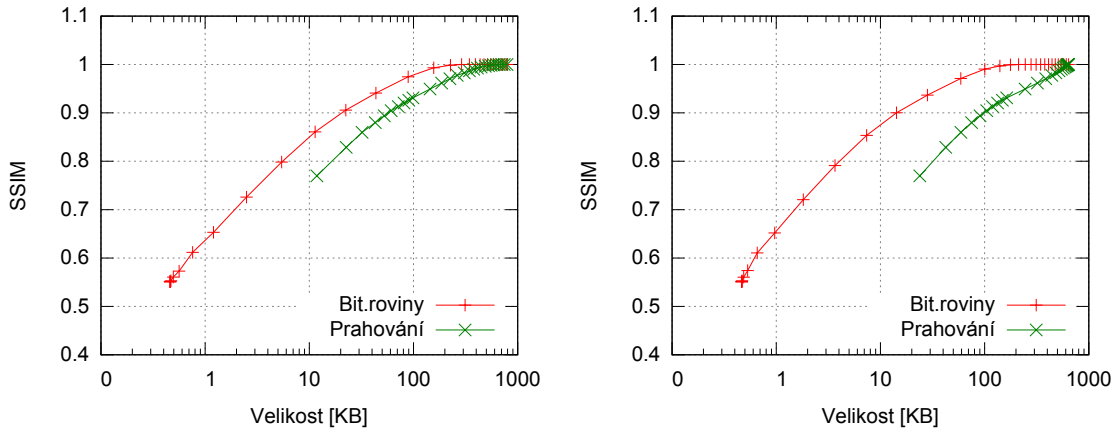
(a) Kvalita určená počtem bitových rovin

(b) Kvalita určena prahováním koeficientů

Obrázek 5.3: Porovnání algoritmů EZW a SPIHT. Použita vlnková transformace na obrazu Lena.

Určení kvality

Knihovna určuje výslednou kvalitu obrazu pomocí dvou metod. První metodou určující kvalitu je tvrdé prahování zadané procentem koeficientů, které mohou být nenulové. Druhým způsobem určení kvality je nastavení konečného prahu bitové roviny pro algoritmy kódovaní koeficientů. V obou případech algoritmy EZW a SPIHT kódují postupně po bitových rovinách. Porovnání těchto dvou metod je zobrazeno na grafech 5.4 pro kódovací algoritmy SPIHT a EZW. Z výsledků porovnání vychází jako lepší varianta určení kvality pomocí bitových rovin u obou algoritmů.



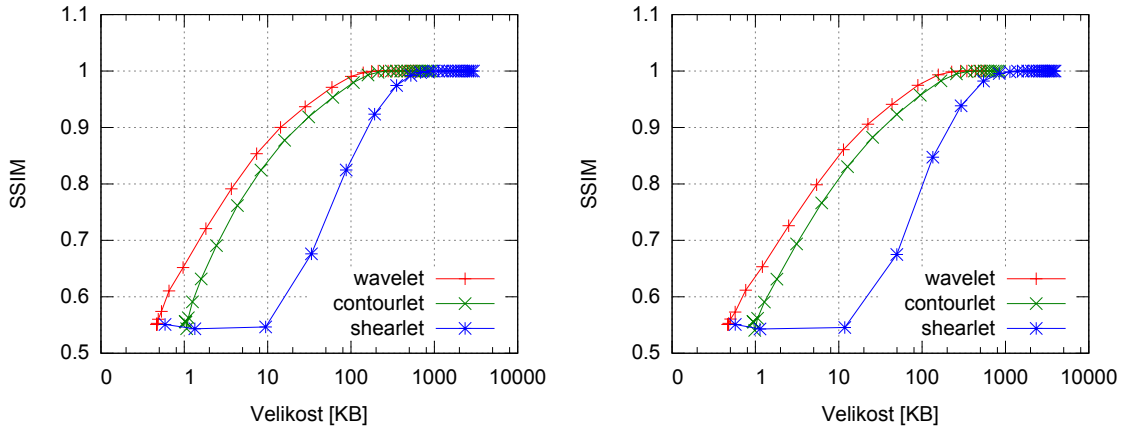
(a) Algoritmus EZW

(b) Algoritmus SPIHT

Obrázek 5.4: Porovnání způsobu určení kvality. Použita vlnková transformace na obrazu Lena.

5.1 Porovnání transformací

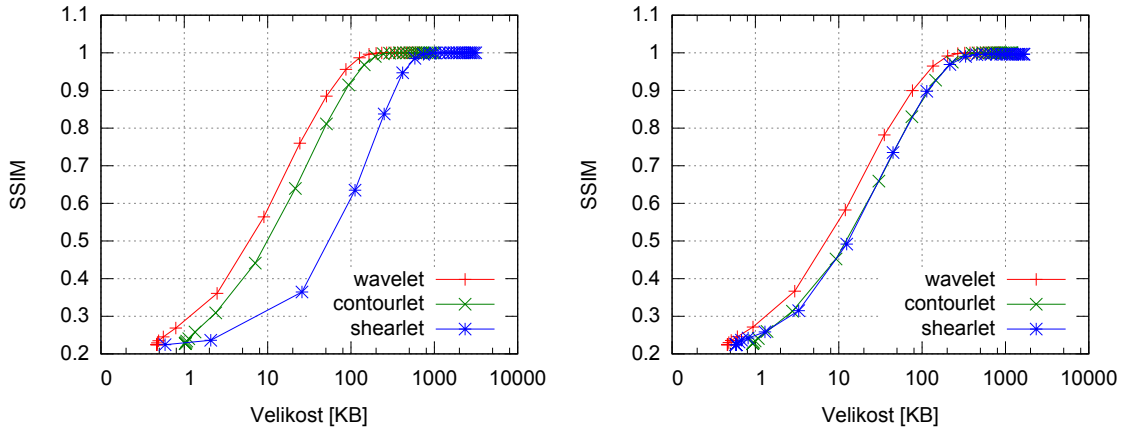
S ohledem na výsledky v grafech 5.4 bude v dalším porovnání uvedeno pouze určení kvality pomocí bitových rovin. Porovnání transformací je nejprve provedeno na obrazu 5.1(a), který představuje přirozený obraz fotografie. Výsledky jednotlivých transformací jsou k vidění v grafech 5.5.



(a) Algoritmus SPIHT (b) Algoritmus EZW

Obrázek 5.5: Porovnání transformací s algoritmy EZW a SPIHT na obraze Lena.

Z grafů 5.5 lze odvodit tři tvrzení pro tento typ obrazu. Kombinace algoritmu SPIHT a určení kvality pomocí bitové roviny dosahuje lepších výsledků než s EZW. Druhým poznatkem je, že konturletová a shearletová transformace dosahuje horších výsledků oproti vlnkové a konturletová transformace dosahuje lepších výsledků než shearletová, kromě případů velké ztrátové komprese s algoritmem EZW. Nakonec shearletová transformace dosahuje lepších výsledků s algoritmem EZW než se SPIHT.

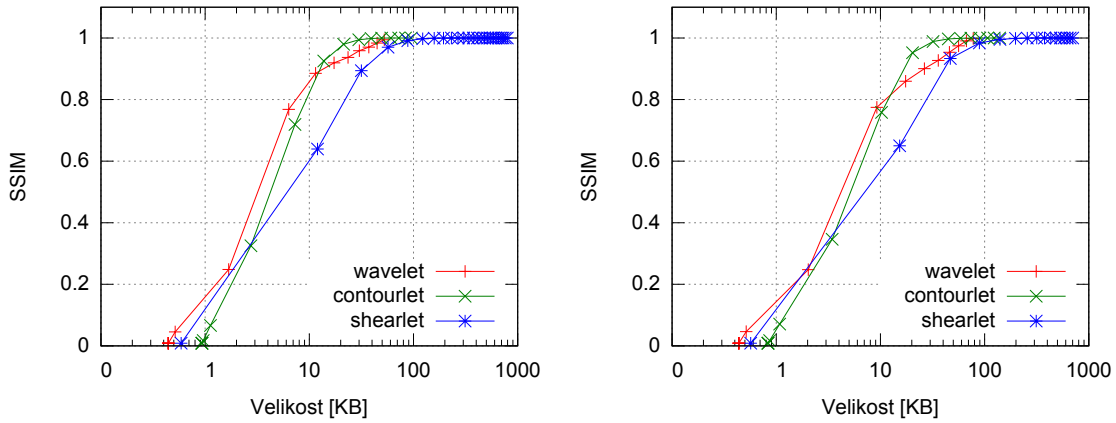


(a) Algoritmus SPIHT (b) Algoritmus EZW

Obrázek 5.6: Porovnání transformací s algoritmy EZW a SPIHT na obraze Baboon. Kvalita určena počtem bitových rovin.

Podobných výsledků je dosaženo i u poměrně vysokofrekvenčního obrazu 5.1(b). Výsledný průběh komprese pro algoritmy EZW a SPIHT je vidět v grafu 5.6. Horší výsledky konturletové a shearletové transformace v obou předchozích porovnání mohou být ovlivněny implementovaným omezením na stejný počet směrů na všech úrovních rozkladu.

Zatím vycházela vlnková transformace jak nejlepší, ale při spuštění komprese (a následné dekomprese) na obraz 5.1(c) bylo dosaženo odlišných výsledků v grafech 5.7. Konturletová transformace nyní předčila vlnkovou. Shearletová transformace zde opět není moc efektivní. Tyto výsledky naznačují, že vhodnějším typem obrazu pro konturletovou transformaci budou obrazy s hodně křivkami a menší stejnosměrnou složkou.



(a) Algoritmus SPIHT (b) Algoritmus EZW

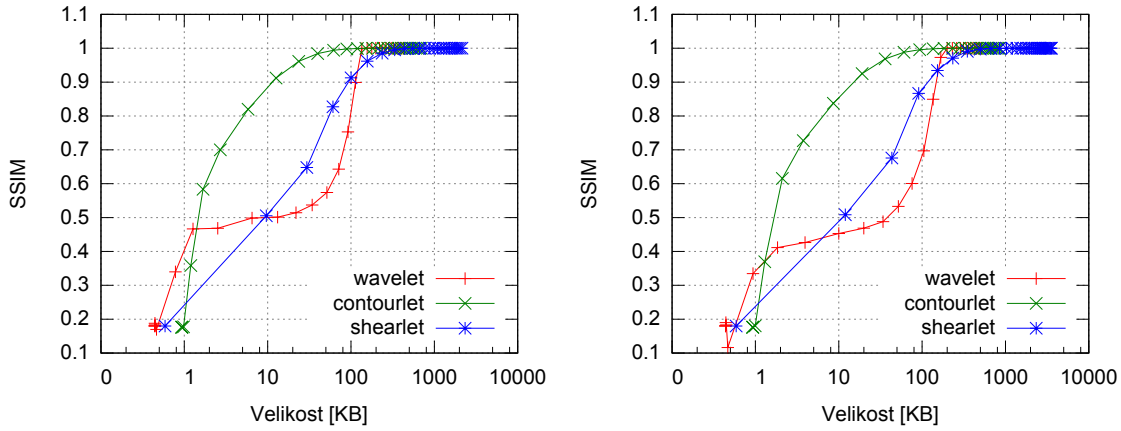
Obrázek 5.7: Porovnání transformací s algoritmy EZW a SPIHT na obraze Zoneplate. Kvalita určena počtem bitových rovin.

Po úvaze nad praktickým využitím této vlastnosti, jsem přistoupil k otestování dvou obrazů otisku prstu. První obraz 5.8(a) představuje jasně zřetelný otisk. Takovýto obraz je prakticky velice obtížné sejmout a je dosažen zpravidla až dodatečnou úpravou. Mnohem realističtější je otisk na obrázku 5.8(b), který obsahuje mnohem více šumu.

(a) Fingerprint1 (b) Fingerprint2

Obrázek 5.8: Příklady upraveného a neupraveného obrazu otisku prstu.

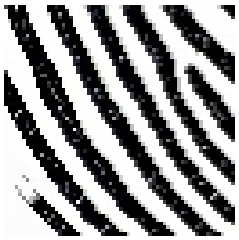
Pro jasně čitelný otisk v obrazu 5.8(a) byly naměřeny výsledky zobrazené grafy 5.9. Z těchto grafů jsou vidět nedostatky vlnkové transformace a naopak přednosti implementované konturletové, ale i shearletové, transformace. Je to dáno tím, že vlnková transformace oproti konturletové a shearletové hůře reprezentuje rysy ve vysokých frekvencích.



(a) Algoritmus SPIHT (b) Algoritmus EZW

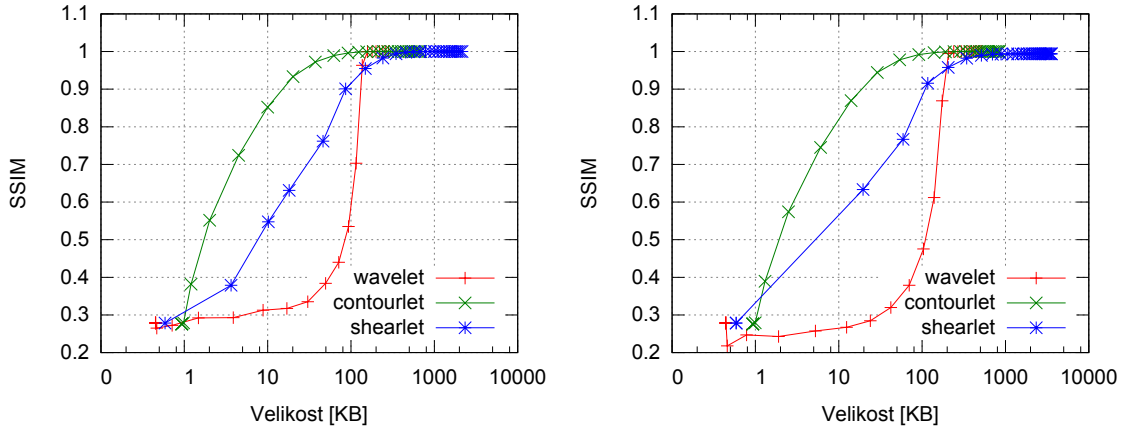
Obrázek 5.9: Porovnání transformací s algoritmy EZW a SPIHT na obraze Fingerprint1. Kvalita určena počtem bitových rovin.

V grafech 5.9 se objevuje u vlnkové transformace nerovnost při vysoké ztrátovosti komprese. Tento skok je dán rychlým získáním obrysů křivek, ale následně šumu (viz obrázek 5.10) v těchto křivkách je obtížně odstraňován.



Obrázek 5.10: Zobrazení šumu v křivkách obrazu otisku při kvalitě.

Předchozí výsledky byly založeny na jasně čitelném otisku, který lze získat velice obtížně, nebo použitím úpravných metod pro zvýraznění rysů a potlačení šumu. V praxi jsou snímány otisky spíše podobné tomu na obrazu 5.8(b). Výsledky v grafu 5.11 ještě dále zobrazují výhody konturletové a shearletové transformace nad vlnkovou. S přidaným šumem jednotlivé výsledky konturletové transformace poněkud poklesly. Shearletová transformace s algoritmem SPIHT v grafu 5.11(a) naopak dosahovala zhruba stejných výsledků jako u upraveného otisku.



(a) Algoritmus SPIHT (b) Algoritmus EZW

Obrázek 5.11: Porovnání transformací s algoritmy EZW a SPIHT na obraze Fingerprint2. Kvalita určena počtem bitových rovin.

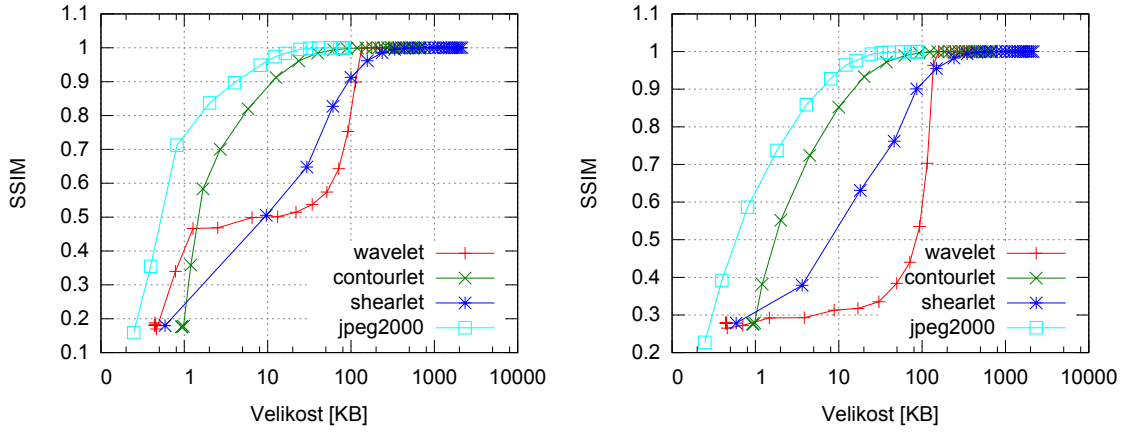
(a) Vlnková transformace (b) Konturletová transformace (c) Shearletová transformace

Obrázek 5.12: Ukázky komprimovaných obrazů algoritmem SPIHT na velikost 10KB.

Z předchozích grafů lze odvodit, že kombinace kódovacího algoritmu SPIHT s určením kvality pomocí bitových rovin dosahuje lepších výsledků než s algoritmem EZW u všech tetovacích obrazů. Z tohoto důvodu budou v další části pouze uvedeny výstupy dosažené za pomoci kombinace algoritmu SPIHT s kvalitou určenou počtem bitových rovin.

5.2 Porovnání s JPEG 2000

Vlnková transformace je použitá v řadě formátů. Jako jejich zástupce byl vybrán formát JPEG 2000 využívající vlnkovou transformaci a pro kódování koeficientů používá algoritmus EBCOT. Jako referenční implementace formátu JPEG 2000 je použit softwarový balík Kakadu [1] od autora kódování EBCOT. Výsledné porovnání pro retušovaný a původní obraz otisku jsou zobrazeny v grafech 5.13. Program pro uložení do formátu JPEG 2000 využíval určování kvality podle počtu bitů na pixel.



(a) Obraz upraveného otisku (b) Obraz neupraveného otisku

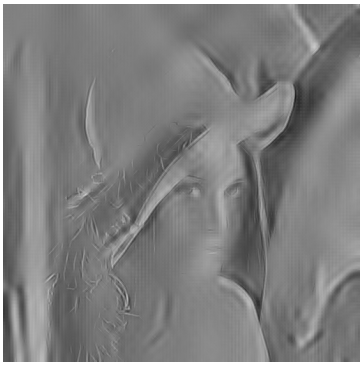
Obrázek 5.13: Porovnání formátu JPEG 2000 softwaru Kakadu a implementovaných transformací s algoritmem SPIHT a kvalitou určenou počtem bitových rovin.

Formát JPEG 2000 vyšel z porovnání jako vítěz, který sice používá vlnkovou trans-

formaci, ale je u něho použit také algoritmus EBCOT na rozdíl od implementovaného SPIHT. V diplomové práci [31] byla prokázána lepší efektivnost algoritmu EBCOT. Software Kakadu je velice dobře propracovaný a optimalizovaný, například kvantizace, oproti implementované, probíhá na každé úrovni s jiným krokem a dokonce se krok liší i mezi některými směry.

Artefakty

Při ztrátové kompresi jsou v obrazech tvořeny různé artefakty. Transformace nahrazuje vstupní obraz hodnotami podobnosti k jednomu nebo více referenčním signálům (vlnky). Ztrátová komprese typicky odstraňuje informace od nejjemnějších rysů. Při zpětné dekompresi se na jejich místech více projeví referenční signál a navíc ztráta vysokofrekvenčních složek způsobuje rozmazání ostrých hran. Na obrázcích 5.15 jsou zobrazeny přiblížené dekódované obrazy, jenž mají podobnost k původnímu obrazu zhruba 0,7 podle metriky SSIM.



Obrázek 5.14: Zachování ostrosti hran shearletovou transformací. SSIM= 0,55

Vlnková transformace, ať už implementovaná (obrázek 5.15(c)) nebo ze softwarového balíku Kakadu (obrázek 5.15(b)), je poměrně rozmazaná a vytváří na křivce nerovnosti.

Konturletová transformace na obrázku 5.15(d) si zachovala o trochu více ostrých hran, neboť výstupní velikost je o něco větší než u vlnkových variant. Artefakty jsou zde tenké ostré čáry, jak je vidět na stínu zhruba ve středu.

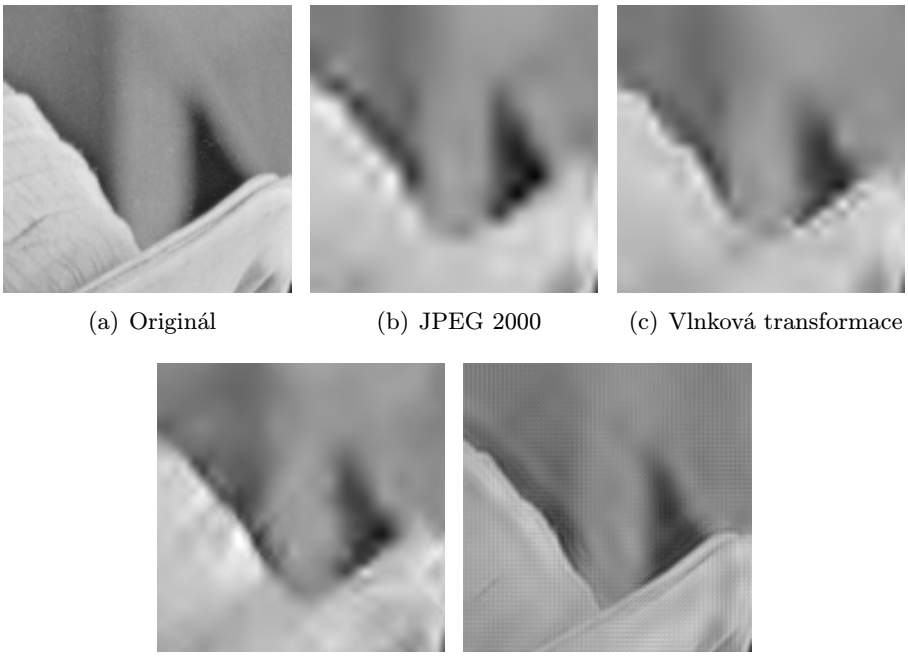
Komprimovaný obraz z shearletové transformace, detail na obrázku 5.15(e), se velice liší oproti ostatním transformacím. Ostrost hran je dána ještě větší velikostí souboru při stejné kvalitě. Shearletová transformace si dobře uchovává ostrost hran, ale v obrazu se objevují mřížkové artefakty.

Při postupném zvyšování ztrátovosti komprese lze zpozorovat u konturletové a vlnkových transformací postupné rozmazávání obrazu. Shearletová transformace si i při vysoké ztrátovosti zachovává ostrost největších hran v obraze, viz obrázek 5.14. Ztrátovost u této transformace je spíše docílena ztrátou méně významných hran, na obraze například prameny vlasů hlavně na levé straně nejsou k rozeznání oproti původnímu obrazu 5.1(a).

5.3 Souhrn srovnání

Jednotlivé transformace (kromě vlnkové) obsahují určitou míru redundance. Konturletová transformace obsahuje až 33 % díky použité Laplaceově pyramidě. Shearletová transformace obsahuje mnohem větší redundanci. Na každé úrovni rozkladu jsou vytvořeny čtyři matice (platí pro implementovanou variantu, teoreticky je redundance ještě vyšší), takže pro například tří úrovňovou dekompozici matice o rozměrech $N \times N$ je vytvořeno $4N^2 + 4(N/4)^2 + 4(N/16)^2$ směrových matic.

První srovnání bylo provedeno na algoritmy kódování koeficientů EZW a SPIHT (grafy 5.3). Z těchto grafů vyplývá, že při určení kvality počtem bitových rovin vychází jako lepší varianta SPIHT. Ovšem při určení kvality tvrdým prahováním koeficientů (zakódované



(a) Originál (b) JPEG 2000 (c) Vlnková transformace

(d) Konturletová transformace (e) Shearletová transformace

Obrázek 5.15: Přiblížené artefakty transformací a JPEG 2000 při kvalitě přibližně SSIM=0,72.

všechny bitové roviny) dosahoval lepších výsledků naopak algoritmus EZW. Následné porovnání obou typů určení kvality na obou algoritmech (grafy 5.4) ukázalo jako lepší variantu omezení počtu bitových rovin.

Další srovnání jednotlivých transformací proběhlo nejdříve na přirozených obrazech fotografií Lena a Baboon, kde se projevila jako lepší varianta vlnková transformace, oproti konturletové a shearletové. U dalšího obrazu Zoneplate již tomu nebyla pravda a konturletová transformace předčila v určitém intervalu vlnkovou. Shearletová transformace se znatelně přiblížila vlnkové.

Poslední dvojicí testovaných obrazů byly otisky prstu. V prvním otisku bylo minimum šumu (typicky po dodatečné úpravě). V tomto porovnání již konturletová transformace znatelně překonala vlnkovou. Shearletová transformace taktéž v určitém intervalu překonala vlnkovou. U druhého otisku prstu obsahující více šumu dopadla konturletová i shearletová transformace zhruba stejně, ale vlnková dosahovala ještě horších výsledků.

Poslední porovnání bylo provedeno znovu na obou obrázcích otisků. Uvedeny pouze výsledky s algoritmem SPIHT, který se projevil o něco lépe než algoritmus EZW. Tentokrát ale byly dosažené výsledky porovnány s formátem JPEG 2000 ze softwaru Kakadu, který dosáhl ještě lepších výsledků, než kterákoliv implementovaná transformace. Z těchto výsledků lze vyvodit, jak je důležité další zpracování koeficientů transformace. Hlavně je to vidět v porovnání implementované vlnkové transformace s Kakadu implementací.

Kapitola 6

Závěr

Tato technická zpráva se skládá ze čtyř kapitol (vyjma úvodu a závěru). Kapitola *Komprese obrazu* uvádí do problematiky komprese obrazu a vysvětluje další pojmy ohledně principů komprese, barevných modelů a způsobů měření kvality, které nejsou přímo tématem práce, ale jsou potřebné v následujících postupech.

V další kapitole *Diskrétní transformace* jsou uvedeny nezbytné pojmy měřítkové funkce a multirozkladu. Následuje podrobný popis diskrétní vlnkové transformace ve 2D prostoru. Konturletová transformace je popsána pomocí konturletové banky filtrů, jenž byla složena z Laplaceovy pyramidy a směrových filtrů. Směrová banka filtrů je definována pomocí quincunx filtrů a shearing operátorů. Shearletová transformace je popsána podobně multirozkladem a tvorbou samotných shearlet. Dále jsou popsány algoritmy pro kódování koeficientů EZW a SPIHT. Okrajově je zde zmíněn EBCOT algoritmus.

První z dvou zbývajících kapitol *Implementace* začíná popisem návrhu struktury vzniklé knihovny. Kapitola dále pokračuje detailním popisem struktury implementace po jednotlivých logických blocích. Poslední kapitola vlastního obsahu této práce *Srovnání* obsahuje postup srovnávání od zvolených algoritmů pro kódování koeficientů a jednotlivých způsobů měření kvality, přes porovnání mezi jednotlivými transformacemi, až po srovnání s formátem JPEG 2000 včetně porovnání vzniklých artefaktů.

Zhodnocení výsledků

Dosažené výsledky nepředčily implementaci Kakadu formátu JPEG 2000. Nicméně ale porovnání implementované vlnkové transformace s Kakadu dokazuje významnost dalšího zpracování výstupních koeficientů z transformací. Dalším výsledkem bylo nalezení třídy obrazu, u které implementovaná konturletová, ale i shearletová, transformace dosahuje lepších výsledků v porovnání s vlnkovou transformací, která byla dále totožně zpracována. Výsledky u přirozených obrazů Leny a Baboon mohou být ovlivněny implementačními omezeními konturletové a shearletové transformace. Jedná se pevně daný počet různých směrů na každé úrovni rozkladu.

Další srovnání je mezi algoritmy EZW a SPIHT společně se dvěma způsoby ovlivnění výsledné kvality komprese tvrdým prahováním koeficientů a určení počtu kódovaných bitových rovin. Algoritmus EZW sice byl efektivnější než SPIHT při určení kvality prahováním koeficientů, ale další srovnání ukázalo lepší efektivnost pro kvalitu danou počtem bitových rovin. Tento způsob určení kvality byl také lépe zpracován algoritmem SPIHT.

Poslední srovnání zahrnovalo vytvořené artefakty při dané kvalitě dekódovaného obrazu. Vlnková transformace i výsledky z Kakadu softwaru vytvářeli ozubení na souvislých

křivkách. Implementovaná vlnková transformace sice vytvořila ostřejší ozubení než výstup z Kakadu, ale bylo méně rozmazané okolí. Konturletová transformace vytvořila podstatně menší ozubení. Namísto toho se v obraze vyskytují přímé tenké artefakty. Shearletová transformace si zachovala nejostřejší hrany, které zůstali dobře zachované i při velké kompresi.

Vlastní přínos

V prvé řadě lze považovat za vlastní přínos tato technická zpráva, která shrnuje kompresi obrazu. K tomu jsou zde přehledně popsány transformace vlnková, konturletová a shearletová.

Hlavním přínosem je ale implementovaná knihovna a aplikace ji využívající. Knihovna poskytuje dostatečně modulární řešení kompresního postupu pro snadné srovnání různých postupů v daném bloku komprese. Hlavním cílem bylo porovnat tři uvedené transformace. Práce ale byla rozšířena o další srovnání dvou implementovaných algoritmů a dvou způsobů určení výsledné kvality. Modularita knihovny si vyžádala úpravu struktury výsledků jednotlivých transformací na uniformní tvar, který musel být dále zohledněn v algoritmech EZW a SPIHT. Tyto algoritmy byly dále upraveny zobecněním, jelikož byly převzaty z implementace pro strukturu koeficientů klasické vlnkové transformace.

Dalším přínosem je vytvořená demonstrační aplikace, která využívá ke kompresi a následné dekompresi implementovanou knihovnu. Druhou funkcí tohoto nástroje je porovnání dvou obrazů na základě metriky SSIM.

Další vývoj

Oblast komprese obrazu neustále hledá nové způsoby reprezentace obrazu za účelem nejen komprimace, ale i jiného zpracování, jako například odstranění šumu a zlepšení kvality daného obrazu. V této práci byly srovnány dané transformace z pohledu komprese. Při využití k jinému účelu mohou dosahovat podstatně jiných výsledků. Z pohledu komprese, ale další pokračování může nabývat dvou směrů.

První možným způsobem pokračování je zkoumání dalších transformací. Kromě známých jako jsou kosinová transformace, je možné porovnat i Noiselet a Bandelet transformaci. Další možností je použití kriticky vzorkovaných konturlet [19]. Druhý možný směr pokračování, který by mohl značně vylepšit efektivnost komprese, je zakomponování jiného algoritmu kódování koeficientů jako například EBCOT, který by lépe zohlednil oblasti ve vysokých frekvencí ve výsledném bitovém toku. Samozřejmě může být další postup kombinací předešlých dvou směrů.

V každém případě lze také dosáhnout zlepšení i při určování kvality na daný bitový tok různými metodami. Od jiného způsobu kvantizace, jako proměnného kroku v jednotlivých úrovní rozkladu. V další řadě se lze zaměřit na poměr velikosti barevných složek ve výstupním toku, kde lze změnit barevné složky zmenšit ve prospěch jasové složky. Z toho vyplývá i možnost hlouběji prozkoumat použití jiných barevných modelů, neboť v této práci byl použit pouze model $\mathrm{YC}_B\mathrm{C}_R$.

Jako další možnost vývoje knihovny by mohlo být vytvoření kombinace různých transformací pro zpracování frekvenčních oblastí. Například zpracování obrazu transformací, která dobře reprezentuje vysokofrekvenční oblasti, do určitého stupně rozkladu a následné zpracování aproximační matice jinou transformací pro nízkofrekvenční oblast.

Literatura

[1] Kakadu software. 2014, software.
URL http://kakadusoftware.com

[2] OpenCV. 2015, software knihovna.
URL http://opencv.org

[3] Addison, P.: *The Illustrated Wavelet Transform Handbook: Introductory Theory and Applications in Science, Engineering, Medicine and Finance*. Taylor & Francis, 2002, ISBN 9781420033397.

[4] Burt, P.; Adelson, E.: The Laplacian pyramid as a compact image code. *Communications, IEEE Transactions on*, ročník 31, č. 4, 1983: s. 532–540, ISSN 0090–6778, doi:10.1109/TCOM.1983.1095851.

[5] Candes, E. J.; Donoho, D. L.: Curvelets: a surprisingly effective nonadaptive representation of objects with edges. In *Curve and surface fitting*, Saint–Malo: University Press, 2000, ISBN 0826513573, 9780826513571, s. 105–120.

[6] Chai, B.-B.; Vass, J.; Zhuang, S.: Significance-linked connected component analysis for low bit rate image coding. In *Image Processing, 1997. Proceedings., International Conference on*, ročník 2, Říjen 1997, s. 637–640, doi:10.1109/ICIP.1997.638852.

[7] Chappelier, V.; Guillemot, C.; Marinkovic, S.: Image coding with iterated contourlet and wavelet transforms. In *Image Processing, 2004. ICIP'04. 2004 International Conference on*, ročník 5, IEEE, 2004, ISSN 1522-4880, s. 3157–3160, doi:10.1109/ICIP.2004.1421783.

[8] 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.

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

[10] Easley, G.; Labate, D.; Lim, W.-Q.: Sparse directional image representations using the discrete shearlet transform. *Applied and Computational Harmonic Analysis*, ročník 25, č. 1, 2008: s. 25–46, doi:10.1016/j.acha.2007.09.003.

[11] Ebrahimi, F.; Chamik, M.; Winkler, S.: JPEG vs. JPEG 2000: an objective comparison of image encoding quality. In *SPIE Applications of Digital Image Processing*, ročník 5558, Denver, 2004, s. 300–308, doi:10.1117/12.564835.

[12] Frigo, M.; Johnson, S. G.: FFTW. 2014, software knihovna.
URL http://www.fftw.org/

[13] Hamilton, E.: JPEG File Interchange Format. *C-Cube Microsystems*, Září 1992, specifikace formátu v1.2.

[14] Häuser, S.; Steidl, G.: Convex Multiclass Segmentation with Shearlet Regularization. *International Journal of Computer Mathematics*, ročník 90, č. 1, 2013: s. 62–81, doi:10.1080/00207160.2012.688960.

[15] Hsiang, S.-T.; Woods, J.: Embedded image coding using zeroblocks of subband/wavelet coefficients and context modeling. In *Circuits and Systems, 2000. Proceedings. ISCAS 2000 Geneva. The 2000 IEEE International Symposium on*, ročník 3, 2000, s. 662–665, doi:10.1109/ISCAS.2000.856147.

[16] Hussain, R.: C++ Wavelet Libraries. 2011, softwarová knihovna.
URL http://wavelet2d.sourceforge.net/

[17] Jégou, H.; Chappelier, V.; Cayre, F.: Information Theory and Signal Processing Library. 2005–2006, software knihovna.
URL http://libit.sourceforge.net/

[18] Kociołek, M.; Materka, A.; Strzelecki, M.; aj.: Discrete wavelet transform–derived features for digital image texture analysis. In *Proc. of Interational Conference on Signals and Electronic Systems*, Lodz, Poland, 2001, s. 163–168.

[19] Lu, Y.; Do, M. N.: CRISP contourlets: a critically sampled directional multiresolution image representation. In *Optical Science and Technology, SPIE's 48th Annual Meeting*, San Diego, USA: International Society for Optics and Photonics, 2003, s. 655–665.

[20] MacKay, D.: Compression algorithms. 12/02/2004, webová stránka.
URL http://www.cs.toronto.edu/~mackay/itprnn/code/c/compress/

[21] Mallat, S.: *A Wavelet Tour of Signal Processing: The Sparse Way*. Elsevier Science, 2008, ISBN 9780080922027.

[22] Minh, N.: *Directional multiresolution image representations*. Dizertační práce, Citeseer, 2002.

[23] Owen, T.; Hauck, S.: Arithmetic Compression on SPIHT Encoded Images. 2002.

[24] Puchinger, R.: ImShrinker. 05/07/2013, software.
URL https://github.com/renp/imshrinker

[25] Said, A.; Pearlman, W. A.: A new, fast, and efficient image codec based on set partitioning in hierarchical trees. *Circuits and Systems for Video Technology, IEEE Transactions on*, ročník 6, č. 3, 1996: s. 243–250, ISSN 1051-8215, doi:10.1109/76.499834.

[26] Servetto, S.; Ramchandran, K.; Orchard, M.: Morphological representation of wavelet data for image coding. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, ročník 4, 1995, ISSN 1520-6149, s. 2229–2232, doi:10.1109/ICASSP.1995.479920.

[27] Shapiro, J.: Embedded image coding using zerotrees of wavelet coefficients. *Signal Processing, IEEE Transactions on*, ročník 41, č. 12, 1993: s. 3445–3462, ISSN 1053–587X, doi:10.1109/78.258085.

[28] Taubman, D.: EBCOT (Embedded Block Coding with Optimized Truncation): A complete reference. *ISO/IEC JTC1/SC29/WG1 N*, ročník 988, 1998.

[29] Taubman, D.; Ordentlich, E.; Weinberger, M.; aj.: Embedded block coding in JPEG 2000. *Signal Processing: Image Communication*, ročník 17, č. 1, 2002: s. 49–72, ISSN 1522-4880, doi:10.1109/ICIP.2000.899218.

[30] Tian, J.; Wells Jr, R. O.: Embedded image coding using wavelet difference reduction. In *Wavelet image and video compression*, Springer, 2002, s. 289–301, doi:10.1007/0-306-47043-8˙17.

[31] Urbánek, P.: Komprese obrazu pomocí vlnkové transformace. 2013, diplomová práce.

[32] Valens, C.: The Fast Lifting Wavelet Transform. 1999–2004, webová stránka. URL http://polyvalens.pagesperso-orange.fr/clemens/lifting/lifting.html

[33] Valens, C.: EZW encoding. 26/02/2004, webová stránka. URL http://polyvalens.pagesperso-orange.fr/clemens/ezw/ezw.html

[34] Vetterli, M.: Multi-dimensional sub-band coding: some theory and algorithms. *Signal processing*, ročník 6, č. 2, 1984: s. 97–112.

[35] Wang, M.; Xiong, Y.: Embedded quadtree wavelets in image compression. Červenec 12 2005, US Patent 6,917,711.

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

[37] Witten, I. H.; Neal, R. M.; Cleary, J. G.: Arithmetic Coding for Data Compression. *Commun. ACM*, ročník 30, č. 6, Červen 1987: s. 520–540, ISSN 0001-0782, doi:10.1145/214762.214771.