VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

BEZEZTRÁTOVÁ KOMPRESE OBRAZU S VYUŽITÍM VLNKOVÉ TRANSFORMACE

LOSSLESS IMAGE COMPRESSION USING WAVELET TRANSFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **JIŘÍ TUMPACH**
AUTHOR

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

BRNO 2017

Abstrakt

Tato práce se zaměřuje na metody komprese obrazu s využitím vlnkové transformace. Zejména na porovnání klasických separabilních dekompozic obrazu a červeno-černé vlnkové transformace. V teoretické části budou probrány různé metody, které lze využít ke kompresi obrazu, včetně možných výhod a nevýhod. V praktické části bude vytvořena knihovna pro kompresi obrazu. Na jejichž základě budou zhodnoceny různé kompresní postupy od metod vlnkové transformace po prediktory. Na závěr budou porovnány dostupné metody komprese obrazu s vytvořenou knihovnou. Navrhované bezeztrátové kompresní metody jsou vylepšením formátů JPEG 2000 a PNG ve zkoumaných datasetech.

Abstract

This work focuses on wavelet transform and its use in image compression particularly on a comparation between classical tensor product wavelets and new kind of second generation wavelet also known as red-black wavelet transform. Although brief comparison of EBCOT modifications, color transforms, wavelets and predictors are discussed too. A framework for an evaluation of some current methods is constructed and results across different image groups are presented. In addition, C++ library was created. Proposed lossless compression methods are better then JPEG 2000 and PNG.

Klíčová slova

Bezeztrátová komprese obrazu, vlnková transformace, červeno-černá vlnková transformace, rozklad obrazu, transformace barev, prediktor

Keywords

Lossless image compression, wavelet transform, red-black wavelet transform, image decomposition, color transform, predictor

Citace

TUMPACH, Jiří. *Bezeztrátová komprese obrazu s využitím vlnkové transformace.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bařina David.

Bezeztrátová komprese obrazu s využitím vlnkové transformace

Prohlášení

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

.....
Jiří Tumpach
11. května 2017

Poděkování

Děkuji vedoucímu práce Ing. Davidu Bařinovi za spolupráci při vytváření této bakalářské práce. Také děkuji všem autorům, od kterých jsem mohl čerpat cenné informace.

Obsah

Kapitola 1

Úvod

S technickým rozvojem se množství generovaného digitálního obsahu na jednoho člověka neustále zvětšuje. Proto stále vzniká problém, jak tento obsah efektivně uložit. Jednou z možností, jak řešit tento problém, je převedení těchto dat do jiné podoby neboli formátu. Pokud jsou data s tímto formátem účelně méně objemná než originální data, mluvíme o kompresi. Je vhodné, aby tento formát byl co možná nejúspornější z pohledu velikosti komprimovaných dat. Další požadavky mohou být na rychlost převádění formátů a jiné manipulace s těmito daty.

Abychom mohli vytvořit co možná nejvhodnější formát, je důležité znát společné vlastnosti dat, se kterými se manipuluje. Teprve potom lze maximálně využít těchto informací a vytvořit ideálně neredundantní podobu těchto dat, která splňuje požadovaná kriteria.

Tato práce se zaměřuje na kompresi obrazových dat, avšak probraná témata lze aplikovat i v příbuzných oborech. Právě obrázky mají mnoho vlastností, které lze efektivně využít. V následující teoretické části budou rozebrány některé vlastnosti a metody, které je využívají k účelům komprese obrazu. Práce se bude zaobírat především diskrétní vlnkovou transformací, ale zmíním se i o prediktorech a entropických kodérech.

V praktické části budou některé metody implementovány v podobě programu a knihovny v jazyce C++. Uvedené metody porovnám mezi sebou včetně jejich kombinací a zhodnotím jejich vhodnost u různých typů obrázků. Výsledkem budou hlavně posloupnosti technik vedoucích k nejlepším dosaženým výsledkům v dané kategorii a jejich porovnání s aktuálně používanými řešeními.

Kapitola 2

Teoretická část

V následující kapitole budou popsány metody, které byly při konstrukci komprimační knihovny zvažovány nebo přímo implementovány. Je předpokládána čtenářova základní znalost o kompresi, díky které je možné obsáhnout více témat.

2.1 Barevné transformace

Nejběžnější aktuálně používaný barevný model je model RGB. Jedna barva je v tomto modelu zachycena jako tři čísla, která reprezentují postupně červenou, zelenou a modrou barevnou složku [1]. Tyto složky nabývají hodnot v závislosti na barevné hloubce, často to jsou bezznaménková čísla v rozsahu od 0 do 255.

Pro kompresi však má tento model jednu nevhodnou vlastnost: V rámci jednotlivých pixelů jsou hodnoty reprezentující barevné složky příliš moc korelované. V tomto případě lze očekávat podobně vysoké/nízké hodnoty v jednotlivých složkách. I když je možné, že nějaký obrázek v tomto modelu bude dekorelovaný, reálně je situace blíže pravému opaku.

Na běžných obrázcích je tedy vidět obrovské množství velmi tmavých až černých pixelu (RGB: $0, 0, 0$) v protikladu s výrazně světlými (RGB: $255, 255, 255$).

Pro jednodušší implementaci celé kompresní metody je proto vhodné nejprve provést barevnou transformaci do nějakého výhodnějšího barevného prostoru. Nejlépe do takového ve kterém budou složky plně dekorelované. V dalších krocích komprese by se díky tomu mohlo hledět na tyto složky jako na 3 odlišné obrázky. Protože se neukládá takové množství redundance, lze docílit i lepšího výsledného kompresního poměru.

(a) červená

(b) zelená

Obrázky 2.1: Barevné složky modelu RGB a jejich viditelná korelace

2.1.1 Karhunen-Loève transformace

Úplného odstranění korelace lze docílit s pomocí KLT (PCA) [23, 25]. Tato lineární transformace umožní úplnou dekorelaci barevných komponent. Lze tak učinit buď za základě reprezentativního vzorku, nebo pro každý obrázek vypočítat ideální bázi zvlášť. Nicméně pro praktické účely bezeztrátové komprese obrázku je nevhodná hned z několika důvodů:

- Je výpočetně náročná.
- Její výstup jsou reálná čísla.
- Může způsobovat těžce komprimovatelný šum.

Právě reálná čísla se v počítačích reprezentují pomocí relativně velkého množství bitů. Při výpočtech s touto číselnou reprezentací navíc dochází k chybám, které v bezeztrátové kompresi mohou způsobovat značný problém. I kdybychom použili jakousi celočíselnou aproximaci, mohl by vznikat problém s rozsahem těchto hodnot a především s jejich plynulým navazováním.[1] Z těchto informací lze odvodit, že je potřeba hodnotit barevné prostory hned z několika měřítek. Zde je několik příkladů:

- Výpočetní náročnost
- Průměrné snížení korelace
- Změna oboru hodnot pixelů
- Navazování hodnot podobných (často vedlejších) pixelů

Kromě KLT existují transformace do jiných barevných prostorů. Snaží se být kompromisem mezi dříve zmíněnými kritérii, to bohužel znamená i to, že výstup není ideální.

2.1.2 YUV

Lidské oko obsahuje dva druhy receptorů, které umožňují vidět – tyčinky a čípky [16]. Tyčinek je více a jsou více citlivé, zajišťují vnímání světlosti – černobílé vidění. Čípků jsou tři druhy, umožňují vnímání červené, modré a zelené barvy. Zatímco by se dalo říci, že model RGB je odvozen pouze ze znalostí o barevném vnímání světla, YUV modely lépe zachycují vlastnosti lidského oka.

YUV je skupinové označení pro barevné modely, které zachycují zvlášť jasovou složku vedle dvou barevných [1]. Protože lépe odráží lidské vnímání barev, lze je i lépe aplikovat v kompresních úlohách. Vlastnost těchto modelů lze nejvíce zužitkovat u ztrátových formátů tím, že se vyhradí více bitů pro jas a méně pro barevné složky. V případě modelu RGB lze obdobně využít citlivost různých barev (např. modrá je nejméně citlivá) [16].

V případě bezeztrátových formátů pochopitelně tyto operace nejsou přípustné. Avšak bezeztrátové formáty typu YUV často poskytují snížení korelace, a tak poskytují výsledné zlepšení kompresního poměru [23, 1].

Ve skutečnosti se nejedná přímo o jas, ale technologicky přijatelnější aproximaci viz ukázka 2.2. Tyto převody se často uskutečňují jednoduchým výpočtem obsahujícím pouze základní aritmetické operace.

Pixel se na rozdíl od KLT transformuje vždy stejně, což umožňuje výrazně rychlejší výpočet, ale taky výrazně zhorší dekorelaci výstupu. Dále to znamená, že tyto transformace vykazují různý výkon napříč různými obrázky. Některé jsou lepší pro obrázky zachycující přírodu, další může být lepší pro obrázky z vesmíru.

[1] Navazování hodnot bude využíváno v dalších krocích komprese.

$$Y_1 = \begin{pmatrix} 0,299 \\ 0,587 \\ 0,114 \end{pmatrix}^T \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad Y_2 = \begin{pmatrix} 0,2125 \\ 0,7154 \\ 0,0721 \end{pmatrix}^T \begin{pmatrix} R \\ G \\ B \end{pmatrix} \qquad Y_3 = \left\lfloor \frac{R + 2G + B}{4} \right\rfloor$$

Ukázka 2.2: Různý výpočet jasové složky z modelu RGB

2.1.3 Plně reverzibilní transformace

Pro bezeztrátovou kompresi obrazu je důležité, aby všechny transformace nad daty byly plně obnovitelné do původní podoby. To platí také pro převod do jiného barevného prostoru.

Jako cenu za plnou reverzibilitu a celočíselný výstup často rozšiřují obor hodnot pixelu, takže je nutné jednotlivé barevné složky uložit pomocí typu s dostatečnou rezervou [23]. I přes rozšíření se ukázal výtěžek z několika málo aritmetických operací navíc jako výrazný krok k lepšímu kompresnímu poměru (po aplikaci dalších kompresních metod). Takové transformace jsou např. RCT (JPEG 2000), YCoCg (MPEG) [23, 24, 19].

JPEG 2000 RCT

Tento typ barevné transformace se používá ve standardu JPEG 2000 při bezeztrátové kompresi [23, 24]. Výstup z této transformace lze uložit jako celá čísla v podobě tří hodnot – Y, C_u, C_v. Konkrétní postup výpočtu je uveden v rovnici 2.1, zpětný převod do modelu RGB lze nalézt v rovnici 2.2.

$$\begin{aligned} Y &= \left\lfloor \frac{R + 2G + B}{4} \right\rfloor \\ C_u &= B - G \\ C_v &= R - G \end{aligned} \tag{2.1}$$

$$\begin{aligned} G &= Y - \left\lfloor \frac{C_u + C_v}{4} \right\rfloor \\ R &= C_v + G \\ B &= C_u + G \end{aligned} \tag{2.2}$$

Nevýhoda tohoto prostoru spočívá v tom, že je nutné uložit složky C_u a C_v do typů o jeden bit větších než původní barevné složky. Vztah definičního oboru k oboru hodnot je přesněji uveden v rovnici 2.3.

$$\begin{aligned} R, G, B &\in \langle 0, 2^m - 1 \rangle \\ Y &\in \langle 0, 2^m - 1 \rangle \\ Cu, Cv &\in \langle -2^m + 1, 2^m - 1 \rangle \end{aligned} \tag{2.3}$$

2.2 Vlnková transformace

Velmi důležitou vlastností běžných fotografií je to, že blízké pixely vykazují podobné hodnoty. Tento jev se ukázal ještě výraznější po aplikaci některé barevné transformace uvedené v předchozí sekci. Nejjednodušeji tohoto jevu využívají prediktory (kapitola 2.3), které odhadnou pravděpodobnou hodnotu podle okolních pixelů a ukládají pouze rozdíly oproti predikci.

Diskrétní vlnková transformace (DWT) představuje nástroj, který má potenciál tyto vlastnosti lépe využít [14, 18, 8]. DWT převádí signál na jeho částečně časovou a částečně frekvenční podobu. Díky již zmíněným vlastnostem běžných obrázků lze očekávat nízké hodnoty frekvenční složky. S přihlédnutím k tomu že frekvence ani čas se nemusí ukládat explicitně, umožní tato transformace efektivní ukládání signálu s relativně malou entropií.

Druhý pohled na tuto transformaci je v postupném vytváření aproximací v podobě menších obrázků. Vynechané informace – detaily se ukládají zvlášť. S využitím multirozkladu lze aplikovat na tyto aproximace další úrovně této transformace, díky tomu dostaneme jiné detaily dalších úrovní. U těchto detailů lze čekat nízkou entropii, která vede k dobrému kompresnímu poměru.

DWT lze upravovat v závislosti na zpracovávaném signálu velkým množstvím způsobů (hlavně výběr vlnek a dekompozice obrazu) [14]. Problém tedy představuje najít nejvhodnější řešení pro danou aplikaci.

2.2.1 Podobnost s STFT

Diskrétní vlnkové transformace vykazují některé podobné vlastnosti jako krátkodobé Fourierovy transformace (STFT).

Diskrétní Fourierova transformace (DFT) nabízí převod signálu z čistě časové domény do čistě frekvenční [20, 3]. Jenomže podobné pixely lze očekávat pouze v blízkosti daného pixelu, proto by prostý převod celého obrázku na různé frekvence pravděpodobně nepřinesl významné využití tohoto jevu. Logickým řešením je rozdělit obrázek na malá okna a aplikovat DFT pro každé okno zvlášť – diskrétní krátkodobá Fourierova transformace (DSTFT). Tento krok by jistě využil dříve zmíněnou vlastnost obrázku lépe. Protože by mohla mít všechna transformovaná okna podobný výstup, lze očekávat snížení entropie.

DSTFT je možné ještě zdokonalit na diskrétní kosinovou transformaci (DCT), díky které již není potřeba ukládat imaginární čísla reprezentující fázi [14]. Diskrétní kosinová transformace prováděná v oknech je úspěšně použita ve starším, nicméně oblíbeném standardu JPEG. Nicméně má několik nevýhod, které lze s použitím DWT vyřešit lépe. Jednou z nevýhod je to, že díky pevnému oknu nelze zahrnout informace od vzdálenějších pixelů. Dále to může způsobovat prudké rozdíly ve výstupních datech způsobující omezení v použití metod využívajících kontext.

Rozdíl, který přináší WT, spočívá především v časově-frekvenční transformaci, díky které lze optimálněji pokrýt tato data v kontextu principu neurčitosti [8, 14]. A to vyhrazením více bitů (větší rozlišení v čase) pro nízké frekvence a méně pro nižší. Další výhodou je to, že WT existuje i v plně obnovitelné variantě s celočíselnými výstupy [13, 18, 24, 14]. Takže se zdá být velice výhodná nejen pro kompresi obrazu. Pro ztrátovou kompresi navíc umožňuje identifikovat výstupy, které lze postrádat.

2.2.2 Vlastnosti vlnkových transformací

Pro kompresi obrazu se sice spojitá vlnková transformace (CWT) nedá použít, ale její krátké shrnutí by mohlo vysvětlit její diskrétní podobu mnohem lépe. Dovoluji si vynechat několik detailů, protože nejsou předmětem této práce. Nicméně všechny potřebné detaily se lze dočíst v použitých materiálech.

Následuje shrnutí značek používaných ve vzorcích této podkapitoly:

$f(t)$ původní jednorozměrný signál $t \in \mathrm{R}$

$\psi(t)$ mateřská vlnka (mother wavelet)

s měřítko (scale) – rozšiřuje/zužuje mateřskou vlnku

τ posunutí (translation) – posunuje vlnku v čase

$\overline{a}$ komplexně sdružená hodnota k proměnné a

Spojitá vlnková transformace (CWT) je dvojrozměrná funkce (pro jednorozměrný signál), definovaná vztahem (2.4) [14, 8]. Na rozdíl od FT není plně definovaná, ale vyžaduje výběr mateřské vlnky.

Mateřská vlnka (ψ) je funkce implementující pásmovou propust. Díky dilataci mateřské funkce (změna měřítka) lze zachytávat různá frekvenční okna. Zpravidla široké měřítko umožňuje detekci nižších frekvencí a naopak. Nutno podotknout, že až na Shannonovu vlnku se nejedná o dokonalou pásmovou propust. Translace ψ zajišťuje částečnou časovou závislost transformace. Na obrázcích 2.3 lze vidět výběr některých známějších mateřských vlnek.

$$CWT\left[f\left(t\right)\right] = W(\tau, s) = \int_{-\infty}^{+\infty} f(t)\overline{\psi_{\tau,s}(t)}dt \tag{2.4}$$

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}}\psi\left(\frac{t-\tau}{s}\right) \tag{2.5}$$

2.2.3 Diskrétní vlnková transformace

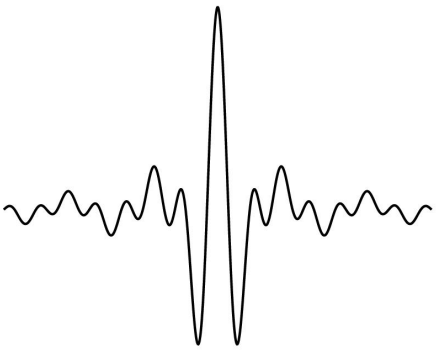
Díky spojitosti v diletaci, měřítku a ve výstupní funkci je počet koeficientů výsledného signálu nekonečný. Řešením je použití diskrétní vlnkové transformace (DWT). Ta se zavádí zdiskretizováním měřítka ($s \to j$) a translace ($\tau \to k$) podle 2.6 [8]. Lokalizace diskrétních τ, s je znázorněná na obrázku 2.4.

$$\begin{aligned} j, k &\in \mathbb{Z} \\ \tau &= k2^j \\ s &= 2^j \\ \psi_{j,k}(t) &= 2^{j/2}\psi(t - 2^j k) \end{aligned} \tag{2.6}$$

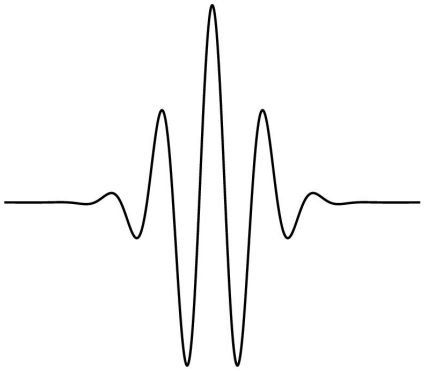
Pro bezeztrátovou kompresi je ale důležité zajištění úplné rekonstrukce signálu a přitom co možná nejnižší redundance. Oba požadavky jsou i po diskretizaci možné. Dokonce při zvolení některých typů vlnek (např. ortogonálních, biortogonálních) lze zajistit úplnou rekonstrukci původních dat při nulové redundanci[2] [14, 10, 8, 28].

Za normálních okolností je nutné provést kompletní transformaci (nekonečné rozšiřování měřítka), aby bylo možné provést transformaci zpětnou. Tento problém je řešen tzv. scaling neboli měřítkovými funkcemi (ϕ). Tyto funkce se chovají jako dolnopropustný filtr a vytváří tak aproximaci signálu na určitém měřítku. Jejich chování lze vidět na obrázku 2.5.

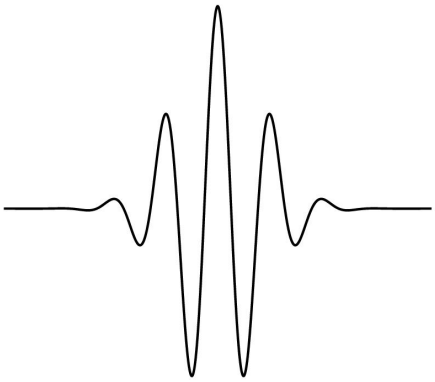
[2] Z pohledu komprese náhodných dat. I po této transformaci je reprezentace obrázků redundantní.



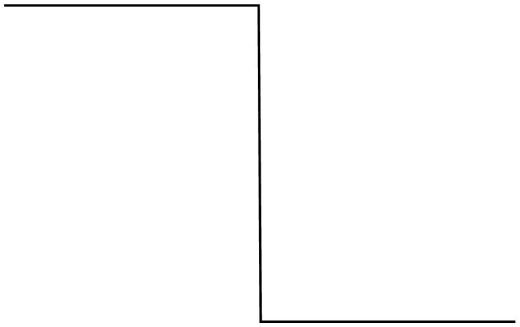
(a) Shanonova vlnka



(b) Morletova vlnka

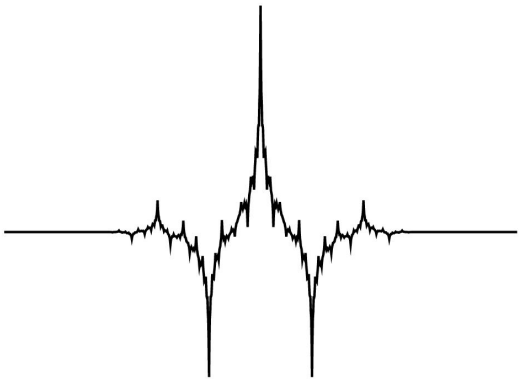


(c) vlnka Mexický klobouk



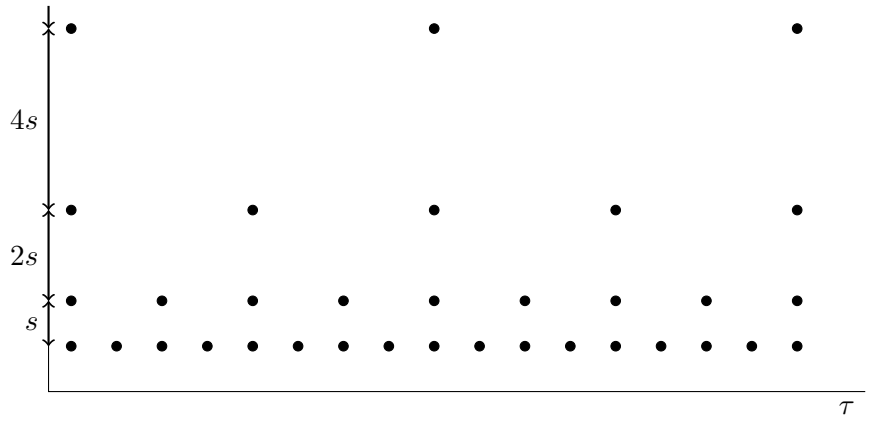
(d) vlnka Haarova/Daubechies 1

(e) vlnka Daubechies 2

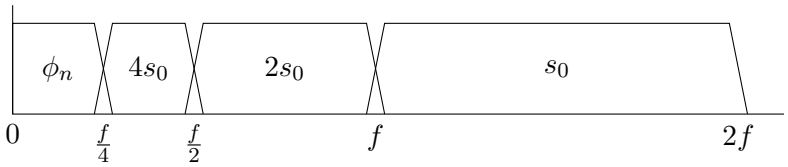


(f) vlnka CDF 5/3

Obrázky 2.3: Výběr některých mateřských vlnek



Obrázek 2.4: Lokalizace vlnek u DWT pomocí diletace a translace



Obrázek 2.5: Zachycení částí spektra vstupního signálu s různými diletacemi vlnek v DWT a měřítková funkce

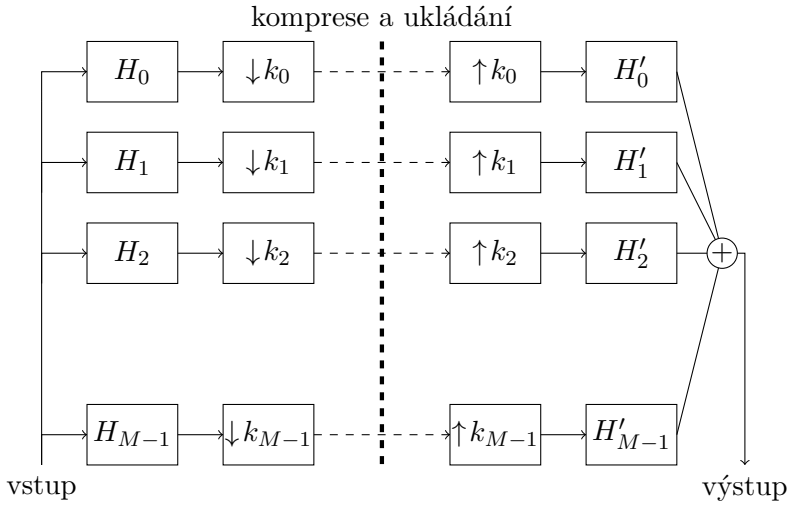
Další podstatné vlastnosti WT jsou tzv. nulové momenty. Zjednoduše lze říci, že pro vlnku s N nulovými momenty je tato transformace nulová v místech, kde lze signál popsat pomocí polynomu $N-1$ řádu. Podrobnější příklady lze vidět i v podkapitolách Haarova transformace a Lineární vlnková transformace (jako příklad s jedním a dvěma nulovými momenty).

2.2.4 Implementace DWT

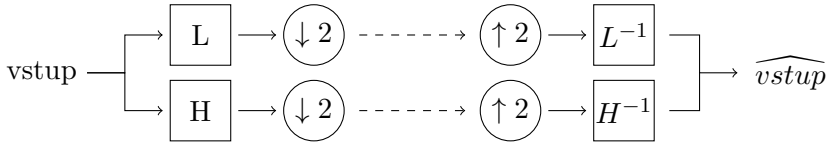
Existuje několik možností, jak implementovat tuto transformaci. Násobení generovaných matic je výpočetně velmi náročné $O(n^2)$ vzhledem k tomu, že pro běžné implementace platí $O(n)$ [7, 28, 14]. Nejen proto se v praxi příliš nepoužívá.

Další možností je vytvořit skupinu vzájemně se doplňujících filtrů se stejným vstupem [10, 14]. Takovéto kupině se říká banka filtrů a je znázorněna na obrázku 2.6. Po provedení konvoluce s filtry jsou výstupní koeficienty podvzorkovány ($\downarrow N$), jinak by byl návrh redundantní. Při reverzní operaci je nutné naopak koeficienty nadvzorkovat – prokládat nulou ($\uparrow N$). Ke všem konstrukčním filtrům $H_0, \ldots, H_{M-1}$ musí existovat vhodné rekonstrukční filtry $H'_0, \ldots, H'_{M-1}$, aby bylo možné obrázek rekonstruovat.

Předchozí konstrukce je náročná na návrh filtrů. Proto je mnohdy jednodušší a výpočetně rychlejší navrhnout banku právě čtyřech filtrů: dvou konstrukčních a dvou rekonstrukčních (obrázek 2.7) [8, 10, 28]. Případné další rozčlenění lze řešit další aplikací transformace na výstup předchozí úrovně podle obrázku 2.8. U konstrukčních filtrů lze v tomto případě mluvit o dolnopropustném případně hornopropustném filtru, i když v celkovém důsledku to tak být nemusí. Podobně jako u obecných banek filtrů i zde dochází ke stejnému podvzorkování a nadvzorkování.



Obrázek 2.6: Banka filtrů



Obrázek 2.7: Kompletní banka včetně rekonstrukční části

2.2.5 Lifting

Výpočet vlnkové transformace lze provést více způsoby (např. násobením generované matice, konvolucí, FFT, algoritmem lifting) [28, 7, 14].

V některých případech je i přes všechny předtím zmíněné výhody nevýhodné ukládat výstupy této transformace, protože neprodukuje celočíselné výsledky [3]

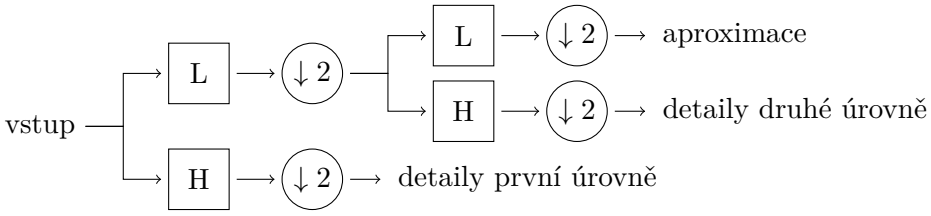
Algoritmus lifting umožňuje relativně lehce aproximovat transformaci a vytvořit její obdobu s celočíselným výsledkem. Navíc umožňuje až 100% nárůst výkonu oproti klasickému postupu. Pro často používané délky filtrů je nárůst zhruba 55 %.

V klasickém filtrování je nutné pro každý výstup spočítat sumu součinů s počtem odpovídajícím délce filtru. Tento algoritmus umožňuje sloučit některé výpočty, a tak snižuje celkovou výpočetní náročnost [13, 28]. Dále umožňuje výpočet in situ (bez dalších alokací paměti) a v některých případech lze využít i instrukce SIMD [14].

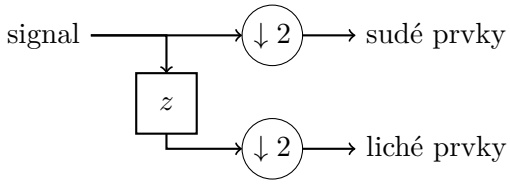
Základní algoritmus lifting představuje 3 základní kroky: split, predict a update [13, 28, 4]. Operace split rozděluje signál na podsignály. Nejčastější způsob je rozdělit liché a sudé vzorky signálu do dvou. Takové WT se označuje jako líná vlnková transformace. Toto chování ilustruji na obrázku 2.9.

Operace predict predikuje následující lichý vstup. Jejím součtem (rozdílem) s tímto vstupem vznikají detaily. Aby aproximace byla validní, musí se ještě upravit na základě předtím vypočteného rozdílu a operace update.

[3] Reálná čísla se hůře komprimují a při ukládání vznikají chyby (IEEE 754). Aby bylo možné přesně uložit některé podíly, musí být bitů mantisy nekonečně mnoho. Řešením je zaokrouhlení na specifikovaný počet bitů, které přináší nepřesnosti. Protože především nejméně významné bity mantisy obsahují zaokrouhlení nějaké neznámé a často i nekonečné posloupnosti, je tato část obtížně predikovatelná a představuje problém [30].

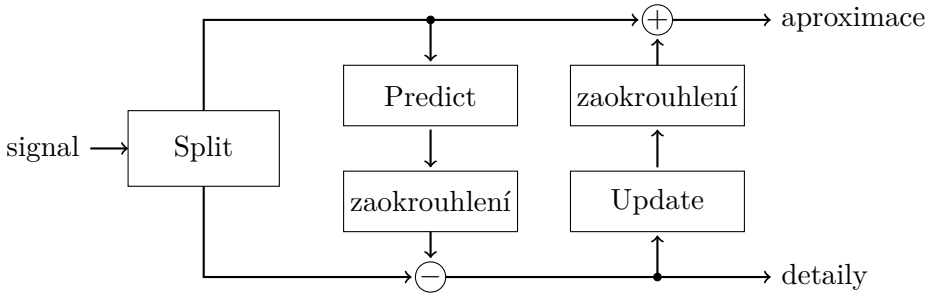


Obrázek 2.8: Ukázka výceúrovňové banky filtrů



Obrázek 2.9: Operace split rozdělí signál na liché a sudé prvky

Protože predict i update nemusí produkovat celá čísla, lze jejich působení upravit zaokrouhlením jejich výsledku. Díky tomuto kroku lze docílit mapování celých čísel na celá. Tok informací v tomto algoritmu ilustruji v 2.10.



Obrázek 2.10: Schéma toku informací u algoritmu lifting

V příkladu 2.7 je ukázka operací predict a update u jednoduché Haarovy vlnky.

V obecném případě nemusí platit toto rozložení na jednu operaci predict a update (jednostupňový lifting), ale na více dvojic podobně pracujících operací, nicméně postup výpočtu zůstává velmi podobný.

Rekonstrukce probíhá obdobně, je důležité převrátit pořadí a provádět reverzní operace. Takže nejprve operace update, která na základě aktuálních detailů odečte výsledek od aproximace. Potom predict přičítá stejně vypočítanou[4] predikci k detailům atd.

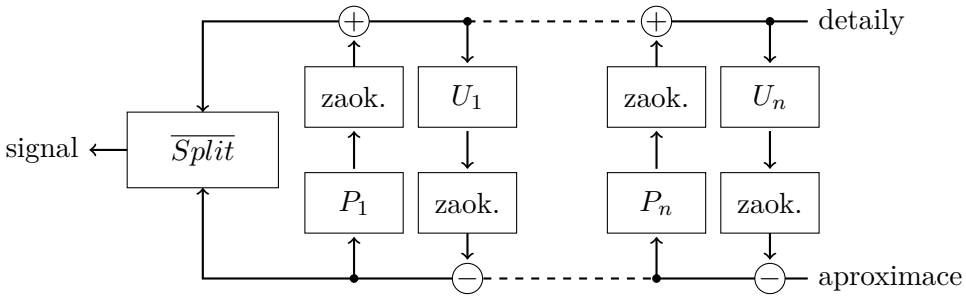
V blokovém schématu se na konci vyskytují bloky pro násobení konstantami. Ty nemusí být celočíselné, a proto je někdy nutné upravit rekonstrukci, aby s tímto počítala.

Bylo dokázáno, že lze jakýkoliv pár konečných filtrů rozložit na předtím zmíněné kroky liftingu [13, 28].

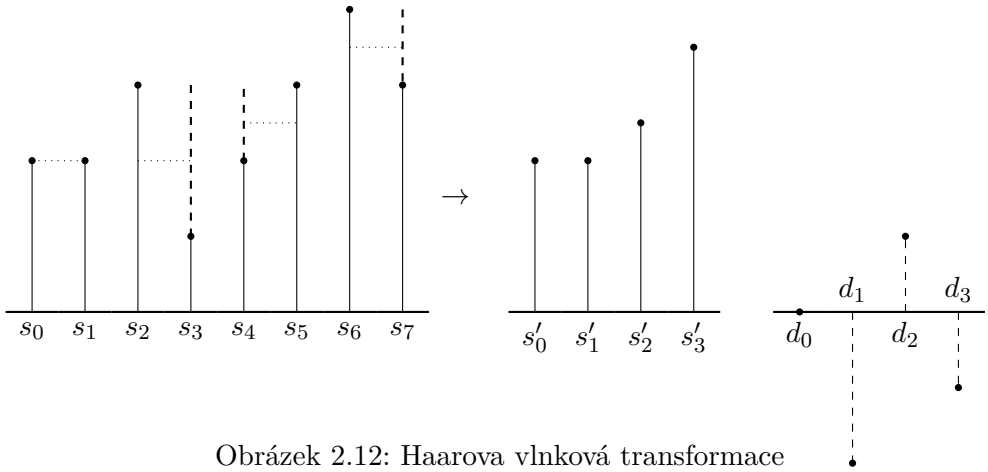
Haarova transformace (db1, D2)

Tato naprosto elementární vlnková transformace zpracovává dvojici bezprostředně následujících vstupních hodnot zvlášť [14]. Výstup z transformace je průměr těchto hodnot

[4] Jako u dopředné transformace.



Obrázek 2.11: Schéma toku informací u vícestupňového rekonstrukčního liftingu



Obrázek 2.12: Haarova vlnková transformace

$(s'_1, s'_2, \dots)$ a jejich rozdíl $(d_1, d_2, \dots)$. Příklad lze vidět na obrázku 2.12, ve kterém jsou průměry označeny tečkovaně, rozdíly čárkovaně.

$$s'_n = \frac{s_{2n} + s_{2n+1}}{2} \qquad d_n = s_{2n} - s_{2n+1}$$

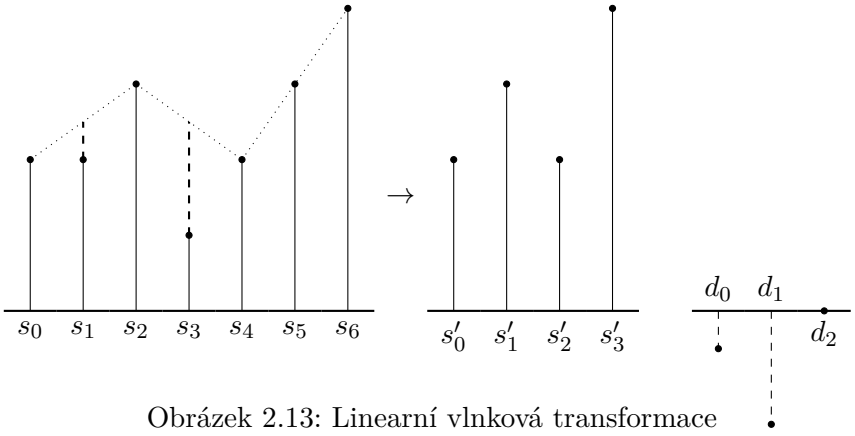
V tomto případě lifting pravděpodobně nepřinese žádné zvýšení výkonu, ale umožní mapování na celá čísla [28, 13]:

$$\begin{aligned} Predict(x) &= x \\ Update(x) &= \left\lfloor \frac{x}{2} \right\rfloor \end{aligned} \tag{2.7}$$

$$\begin{aligned} Split(0) &= a \\ Split(1) &= b \\ detaily(0) &= Split(1) - Predict(0) = b - a \\ aproximace(0) &= Split(0) + Update(0) = a + \left\lfloor \frac{b-a}{2} \right\rfloor \approx \frac{a+b}{2} \end{aligned} \tag{2.8}$$

Lineární vlnková transformace

Haarova transformace byla schopná přesné predikce pouze tehdy, když se obě hodnoty rovnaly [14]. Lineární transformace, jak název napovídá, je schopná navíc předpovídat i lineární růst hodnot (2 nulové momenty).



Obrázek 2.13: Linearní vlnková transformace

Je výhodnější mít v kompresních úlohách transformaci s lepší predikcí. Nicméně příliš dokonalá aproximace by nemusela být výhodná, protože by se mohla interpolovat i na základě vzdáleného šumu a hran.

Lineární transformace využívá liché hodnoty k predikci sudých, a to tak, že lineárně aproximuje sudou hodnotu podle dvou obklopujících lichých hodnot. Výstup tedy představují nezměněné liché hodnoty a rozdíl průměru od skutečné hodnoty sudých pixelů[5].

$$s'_n = s_{2n} \qquad d_n = s_{2n+1} - \frac{s_{2n} + s_{2n+2}}{2}$$

CDF 5/3 a CDF 9/7

Ve standardu JPEG 2000 se obě tyto transformace používají: CDF 5/3 v bezeztrátové kompresi, CDF 9/7 v kompresi ztrátové [24]. Následují možné implementace algoritmů lifting pro tyto vlnky.

Lifting CDF 5/3 [24]:

$$\begin{aligned} d[n] &= d_0[n] - \left\lfloor \frac{s_0[n+1] + s_0[n]}{2} \right\rfloor \\ s[n] &= s_0[n] + \left\lfloor \frac{d[n] + d[n-1] + 0.5}{4} \right\rfloor \end{aligned} \tag{2.9}$$

Lifting CDF 9/7-M [9]:

$$\begin{aligned} d[n] &= d[n] + \left\lfloor \frac{1}{16}\left((s[n-1] + s[n+2]) - 9\left(s[n] + s[n+1]\right)\right) + \frac{1}{2} \right\rfloor \\ s[n] &= s[n] + \left\lfloor \frac{1}{4}\left(d[n-1] + d[n]\right) + \frac{1}{2} \right\rfloor \end{aligned} \tag{2.10}$$

[5] První hodnota je lichá = 1

Lifting CDF 9/7-F [9]:

$$\begin{aligned} d[n] &= d[n] + \left\lfloor \frac{203}{128}\left(-s[n] - s[n+1]\right) + \frac{1}{2} \right\rfloor \\ s[n] &= s[n] + \left\lfloor \frac{217}{4096}\left(-d[n-1] - d[n]\right) + \frac{1}{2} \right\rfloor \\ d[n] &= d[n] + \left\lfloor \frac{113}{128}\left(s[n] + s[n+1]\right) + \frac{1}{2} \right\rfloor \\ s[n] &= s[n] + \left\lfloor \frac{1817}{4906}\left(d[n-1] + d[n]\right) + \frac{1}{2} \right\rfloor \end{aligned} \tag{2.11}$$

Definice všech porovnávaných vlnek lze nalázt v příloze A.2.

Okrajové podmínky

Okraje obrázků představují při provádění konvoluce problém [21, 2, 14]. Standardní rozšíření pomocí nul na okrajích může způsobovat artefakty v jeho transformaci a tím uměle zvyšovat entropii nebo snižovat korelaci. Častým řešením je použít nějaký typ zrcadlového rozšíření.

Nejpraktičtější je rozšíření označené jako zrcadlové rozšíření 2, které je výhodné pro liftingové implementace, protože nemíchá liché a sudé hodnoty a tím zjednoduší zpětný výpočet okrajových vzorků.

0	0	0	a	b	c	rozšíření nulami
a	a	a	a	b	c	replikace okrajové hodnoty
c	b	a	a	b	c	zrcadlové rozšíření
d	c	b	a	b	c	zrcadlové rozšíření 2

Obrázek 2.14: Možné řešení rozšíření hodnot na okraji obrázků

2.2.6 Rozklad obrazu

Protože lifting je určený pro práci s 1D daty, vzniká problém, jak ho použít pro 2D obrázky. Pomocí strategie uplatnění původně 1D transformace na 2D data[6] je možné vytvořit další rozdíly v efektivitě komprese [14, 26]. Přitom je dekompozice nezávislá na zvolené metodě WT.

Jedním ze základních metod dekompozice je tzv. řádková dekompozice. Spočívá v provádění WT v každém řádku odděleně a následným zpracováním jejich aproximací. Tato transformace je sice rychlá, ale její výstup by pravděpodobně nebyl pro účel komprese nejvhodnější, protože se téměř úplně vypustí využívání korelace ve sloupcích.

Existuje velké množství různých typů rozkladů. Vybral jsem proto jen dvě, dle mého názoru nejperspektivnější.

Pyramidová vlnková transformace

Používá se např. v JPEGu 2000 a zdá se být velmi oblíbená a rozšířená [24, 14].

[6] Rozklad lze obecně využít na n-dimenzionální data.

Jeden stupeň této dekompozice vytváří čtyři pásma označená jako: LL, LH, HL a HH. Nejprve se provede řádková transformace pro všechny řádky při vzniku dočasných pásem L a H. Tato pásma se transformují znovu, ale tentokrát ve sloupcích. Takže z L vzniká LL a LH (H → HL a HH). Další úroveň se provádí obdobně s tím rozdílem, že se rozkládá pouze LL pásmo (LL → LLLL, LLLH, LLHL, LLHH). Jednoduchý nákres je možné vidět na obrázku 2.15.

Díky procesu, který tato pásma generuje, vznikají zajímavé vlastnosti: dá se říct, že pásmo LL je aproximace (nízké frekvence v původním obrázku), LH přenáší informaci především o horizontálních hranách, HL o vertikálních a HH o diagonálních hranách.

Zobecnění tohoto způsobu transformace (průchody řádků a sloupců) je také nazýváno separabilní rozklad.

Červeno-černá vlnková transformace

V některých obrázcích by mohl být problémem to, že hrany nejsou z velké části vertikální a horizontální[7]. To může způsobovat zanášení nenulových hodnot do všech pásem[8]. Protože se v entropickém kodéru tato pásma obvykle zpracovávají odděleně, už není možné tyto potenciální informace využít. Já doufám, že by právě tato dekompozice mohla tento problém řešit. Jak bylo naznačeno, jedná se o méně anizotropní, podobně rychlou transformaci, u které nevznikají jiná omezení [26].

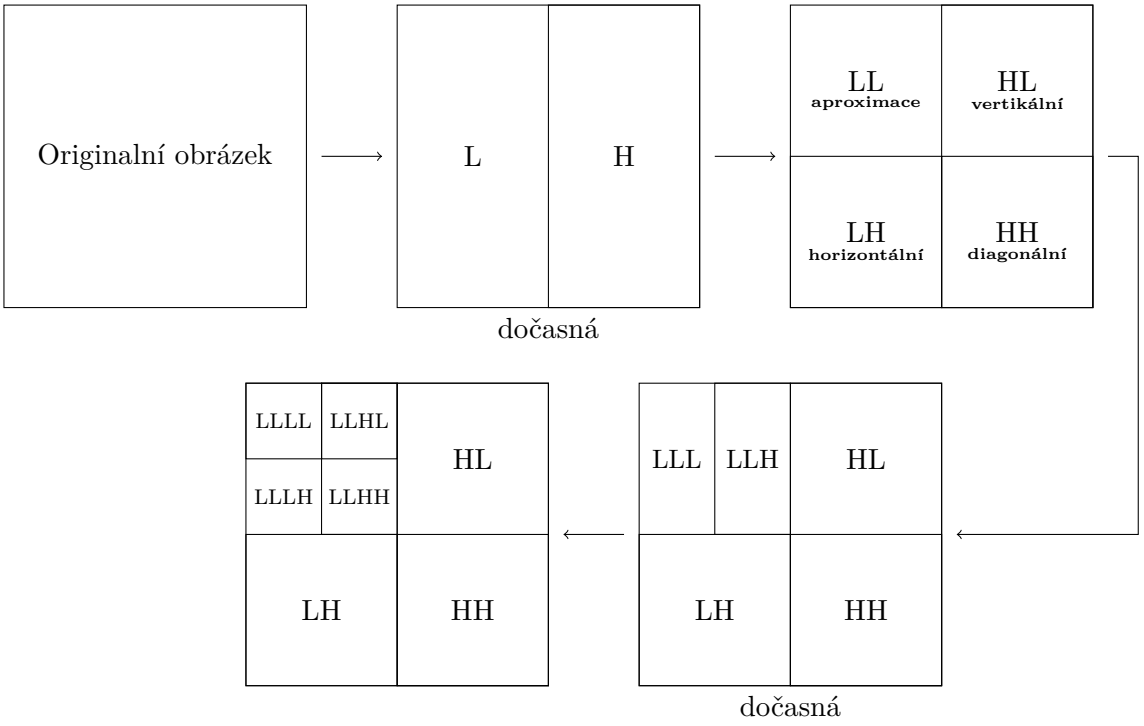
Opět se jedná o metodu o dvou krocích:

Horizontální/vertikální lifting První krok spočívá v pomyslném šachovnicovém rozdělení prvků na červené a černé podle obr. 2.17. Následuje operace predict, která se vypočte z obklopujících červeně označených pixelů. Výsledek okolních operací predit je interpolován[9] a jeho zaokrouhlená[10] hodnota je odečtena od obklopeného černého pixelu (obrázek 2.17a).

Operace update nyní upravuje hodnoty okolních červených pixelů na základě hodnoty v obklopujících černých pixelech (obrázek 2.17b). Černé pixely jsou tímto vypočteny a už nezasahují do dalších kroků – jedná se o výsledné detaily[11]. S červenými pixely se v následujícím kroku pracuje mírně odlišně.

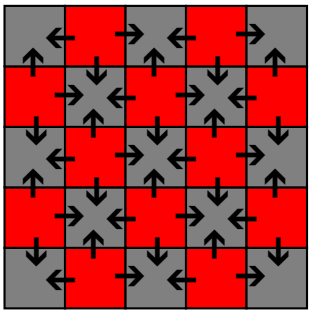
Diagonální lifting Druhý krok zpracovává zbylé červené pixely reprezentující dočasnou aproximaci. Tyto pixely se obdobně rozdělují na modré a žluté. Následují operace predict a update, avšak v diagonálním směru (obrázek 2.18). Tímto způsobem se získávají detaily a aproximace další úrovně. Aproximace může být opět zpracována ve smyslu multirozkladu v dalším horizontálním/vertikálním kroku dle potřeby.

[7] Obrázky z přírody, vesmíru, ... Architektura obvykle patří do druhé kategorie.
[8] Myšleno HL, LH a HH u pyramidové dekompozice obrazu.
[9] Autor uvádí průměr, ale lze si představit i jiné možnosti, např. medián.
[10] Zaokrouhlení je nutné pouze pro zachování mapování na celá čísla. Operace predict se zvlášť zaokrouhlovat nemusí.
[11] V případě vícestupňového liftingu se samozřejmě provedou i další kroky.

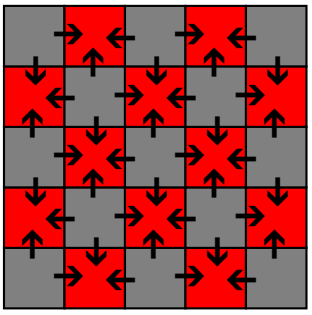


Obrázek 2.15: Dva stupně pyramidové vlnkové transformace

Obrázek 2.16: Zobrazení dat po víceúrovňové pyramidové vlnkové transformaci. Na LH jsou vidět horizontální hrany v HL pásmu vertikální (především u budov v pozadí).

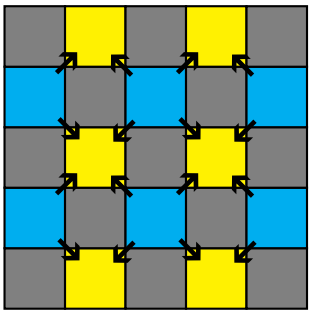


(a) Operace predict

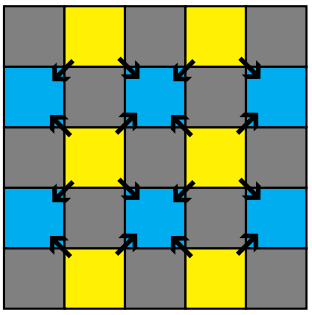


(b) Operace update

Obrázek 2.17: Červeno-černá vlnková transformace – vertikální/diagonální lifting



(a) Operace predict



(b) Operace update

Obrázek 2.18: Červeno-černá vlnková transformace – diagonální lifting

2.3 Prediktory

Prediktory využívají stejnou vlastnost obrázku jako WT a to, že lze očekávat obdobné pixely kolem jiného pixelu. Prediktor je prostředek, který využívá okolí nějakého pixelu pro predikci jeho hodnoty [14]. Predikovaná hodnota je porovnána se skutečnou hodnotou pixelu a v zakódovaném obrázku se uloží pouze rozdíl oproti skutečné hodnotě. Vzhledem k tomu, že je v průběhu dekódování známo celé okolí, se kterým se pixel kódoval, není problém tuto hodnotu znovu získat.

Rozdíl oproti WT je v tom, že prediktory negenerují aproximaci a hlavně se obvykle více aktivně snaží přizpůsobit svou predikci podle detekovaných hran. Protože využívají stejné vlastnosti, omezuje to využití obou těchto technik současně. Nicméně jednou z možností je např. použití prediktorů po WT a to buď na aproximaci nebo na detaily. Pro detaily by vznikl problém, a to že některé algoritmy používané po těchto transformacích opět využívají kontextu, a tak vlastně pracují podobně jako prediktory.

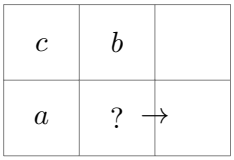
Intuitivně lze očekávat zvyšování frekvence hran v aproximacích vyšších úrovní. Proto by mohlo být výhodné v některé úrovni použít místo dalších WT prediktor. Otázka je, jestli je zavedení další komplexnosti do kompresního postupu výhodné: V této fázi by se mohlo komprimovat výrazně méně informací a celkový rozdíl by tak mohl být zanedbatelný.

WT nemusí být pro všechny obrázky ideálním řešením. Prediktory mohou být vhodnější pro obrázky s převahou čistých hran a ploch. Takové jsou třeba obrázky generované počítačem.

2.3.1 MED prediktor

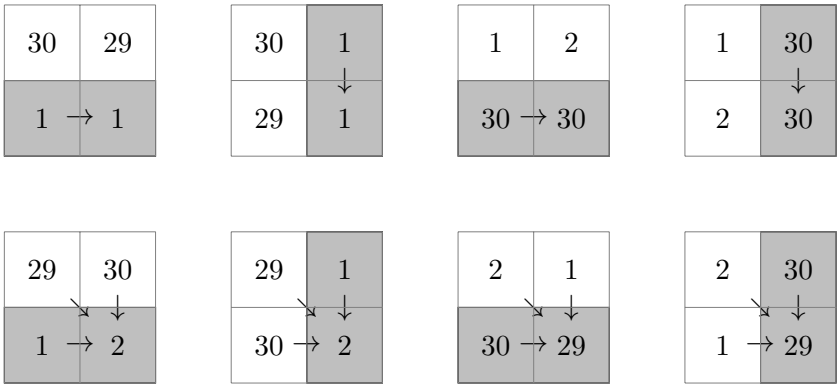
MED (median edge detector) je součástí dnes ne příliš používaného JPEG-LS [29, 11]. Tento prediktor používá několik menších prediktorů, mezi kterými přepíná v závislosti na okolních pixelech. Tímto se pokouší odhadnout hrany a jiné časté prvky obrázků, jak je tomu vidět na obrázku 2.20.

Tento prediktor se mi prakticky ukázal jako velmi výkoný i přes své relativně malé okolí, díky kterému může být více náchylný k šumu.



Obrázek 2.19: Okolí u MED prediktoru

$$predikce = \begin{cases} \min(a,b) & \text{když } c \geq \max(a,b) \\ \max(a,b) & \text{když } c \leq \min(a,b) \\ a+b-c & \text{jinak} \end{cases}$$



Obrázky 2.20: Ukázka MED prediktoru při různých vstupech: Šedá barva znázorňuje hranu, šipky naznačují jak byla vypočtena predikce.

2.3.2 PNG predikce

Známý formát PNG používá predikci založenou na pěti různých prediktorech [22]. Tyto prediktory umožňují predikci více barevných komponent v jednom průchodu[12]. Tato vlastnost může zrychlit všechny operace vázané na tento formát, na druhou stranu značně omezuje konstukci těchto prediktorů. Toto omezení může mít za následek horší kvalitu samotné predikce.

Prediktory none, sub, up a average jsou naprosto základní (linearní potažmo konstantní). Zajímavý je prediktor Paeth: Kontext je stejný jako u MED prediktoru, avšak chování je mírně odlišné. Rozdíl od MED spočívá v tom, že tento prediktor použije vždy jako predikci jednu hodnotu z kontextu.

[12] Predikovaná hodnota je tvořena součtem barevných hodnot, které jsou vhodně bitově posunuty

c	b	
a	$?$ $\rightarrow$	

Obrázek 2.21: Okolí u Paeth prediktoru

$$predikce = \begin{cases} \text{None} & = 0 \\ \text{Sub} & = a \\ \text{Up} & = b \\ \text{Average} & = \lfloor (a+b)/2 \rfloor \\ \text{Paeth} & = \arg\min_x \ |x - (a+b-c)|; \qquad x \in \{a, b, c\} \end{cases}$$

30	30
0	0

30	0
15	0

15	0
30	15

Obrázky 2.22: Ukázka Paeth prediktoru při různých vstupech. Šedá barva znázorňuje detekovanou hranu.

2.3.3 Gradient Adjusted Predictor

Na rozdíl od MED prediktoru, Gradient Adjusted Predictor (GAP) pracuje s drobnějším rozeznáváním hran (ostré, normální, jemné) [31, 11]. Kontext prediktoru je výrazně větší, jak ukazuje obrázek 2.23. Přirozeně se i více zkomplikoval algoritmus výpočtu predikce (viz algoritmus 1).

		NN	NNE
	NW	N	NE
WW	W	$?$	$\rightarrow$

Obrázek 2.23: Okolí používané pro GAP

Výpočet síly hrany je dán:

$$d_h = |W - WW| + |N - NW| + |N - NE|$$
$$d_v = |W - NW| + |N - NN| + |NE - NNE|$$

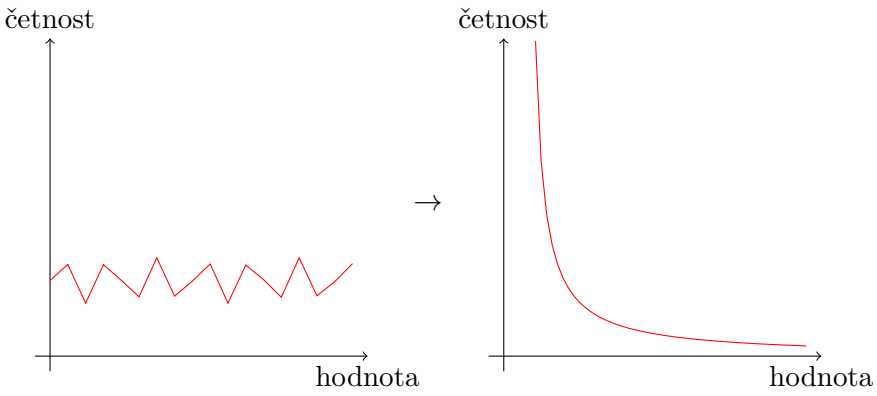
Algoritmus 1: Algoritmus prediktoru GAP

```
if d_v − d_h > 80 then                                  /* ostrá horizontální hrana */
    predikce ← W ;
else if d_v − d_h < −80 then                            /* ostrá vertikální hrana */
    predikce ← N ;
else
    predikce ← (W+N)/2 + (NE−NW)/4 ;
    if d_v − d_h > 32 then                              /* horizontální hrana */
        predikce ← (predikce+W)/2;
    else if d_v − d_h > 8 then                          /* jemná horizontální hrana */
        predikce ← (3predikce+W)/4;
    else if d_v − d_h < −32 then                        /* vertikální hrana */
        predikce ← (predikce+N)/2;
    else if d_v − d_h < −8 then                         /* jemná vertikální hrana */
        predikce ← (3predikce+N)/4;
```

2.4 Entropické kodéry

Předcházející metody vstupní data různě transformovaly – jejich cílem bylo snížit v datech entropii (míru neurčitosti) [14, 24, 27]. Takže nyní lze očekávat velké množství hodnot v malém intervalu, avšak teoreticky by definiční obor mohl významně rozrůst.

Tento problém řeší entropické kódování. Díky těmto metodám lze vyhradit méně bitů velmi častým znakům a více bitů pro málo se vyskytující znaky.



Obrázek 2.24: Příklad vhodné transformace pro entropické kódování

$i \in 0, ..., N-1$	Index znaku i; počet znaků $= N$.	(2.12)
$s_i \in \Sigma$	Znak s_i patří do abecedy Σ.	(2.13)
$p_i = P(s_i)$	Pravděpodobnost znaku s_i je p_i.	
$\Sigma^+ = \Sigma^* - \{\varepsilon\}$	Množina všech konečných řetezců nad Σ.	(2.14)
$g \in \Sigma^+$	g je neprázdný řetezec $\in \Sigma*$	(2.15)
$g = g_0 \cdot g_1 \cdot g_2 \dots g_{M-1}$	g se skládá z M znaků	(2.16)
		(2.17)

Entropické kodéry se snaží transformovat každý znak s_i na posloupnost bitů ideálně o velikosti $-\log_2(p_i)$ (je to teoretický limit).

$$\sum_{x=0}^{M} -P(g_x) \cdot \log_2(P(g_x)) \tag{2.18}$$

Rovnice 2.25: Ideální velikost komprimovaného řetezce

Příklad Předpokládejme kódování řetězce "MINIMUM_MAXIMUM".

Za normálních okolností bude mít tento soubor velikost danou počtem kódovaných znaků a použitým kódováním. Typické kódování v textových souborech (UTF-8, ascii, ...) má znaky o velikosti 8 bitů, tzn. že soubor bude mít $8 \cdot M = 8 \cdot 15 = 120$ bitů.

Druhou možností je omezení abecedy na znaky, které se ve skutečnosti opravdu používají. Zjednodušením, že každý znak bude kódován do celého počtu bitů a že předpokládáme rovnoměrné zastoupení všech znaků, lze docílit velikost $M \cdot \lceil \log_2(N) \rceil = 15 \cdot \lceil \log_2(7) \rceil = 15 \cdot 3 = 45$ bitů.

Některé entropické kodéry umožňují kódovat znak do necelého počtu bitů. Takový soubor by potom mohl mít velikost minimálně $M \cdot \log_2(N) = 15 \cdot \log_2(7) = 15 \cdot 3 \approx 42.11$ bitů.

Když se omezíme na to, že výskyty znaků jsou na sobě nezávislé a splňují pravděpodobnostní model, jenž je dán tabulkou 2.26, lze tento soubor zakódovat minimálně s velikostí

$$\sum_{g_x \in g} -\log_2(P(g_x)) = -6 \cdot \log_2(6/15) - 3 \cdot \log_2(3/15) - 2 \cdot \log_2(2/15) \cdots \approx 36.339 \text{ bitů}$$

Znak	Počet výskytů	Pravděpodobnost výskytů
M	6	$6/15$
I	3	$3/15$
U	2	$2/15$
N	1	$1/15$
A	1	$1/15$
X	1	$1/15$
_	1	$1/15$

Tabulka 2.26: Příklad zastoupení znaků v kódovaném souboru

2.4.1 Kontextové kódování

V předchozí ukázce kódování řetězce "MINIMUM_MAXIMUM" bylo možné všimnout, že kdykoliv program kódoval souhlásku, následovala samohláska nebo mezera a naopak. Kontextové kódování vytváří několik pravděpodobnostních modelů, které vhodně mění v závislosti na okolních znacích, stavu a dalších proměnných [14, 27]. Díky těmto změnám lze významně zvýšit efektivitu kodéru.

Příklad Řekněme, že chceme zakódovat stejné slovo pomocí nějakého kontextového kodéru se dvěma kontexty (k_0, k_1). Kontext k_0 je aktivní na začátku řetězce nebo když předchozí symbol byla mezera nebo samohláska. Kontext k_1 je aktivní, pokud předtím byla kódována souhláska. Symboly kódované s kontextem k_0 budou s vysokou pravděpodobností souhlásky, naopak symboly s kontextem k_1 budou pravděpodobně samohlásky.

Pravděpodobnostní modely kódovaného slova znázorňuje tabulka 2.27. Každý symbol se kóduje na základě pravděpodobnosti v aktuálním kontextu. Výsledná velikost bude tedy minimálně

$$\sum_{(g_x,k)\in G} -\log_2\left(P_k\left(g_x\right)\right) \approx 21.387 \text{ bitů}$$

kde

$$G = (g_0, k_0) \cdot (g_1, R(g_0)) \cdot (g_2, R(g_1)) \cdots (g_{M-1}, R(g_{M-2})) \tag{2.19}$$

$$R(s_i) = \begin{cases} k_1, & \text{když } s_i \text{ je souhláska} \\ k_0, & \text{ostatní případy} \end{cases} \tag{2.20}$$

$$P_k(s_i) = \text{Pravděpodobnost v rámci daného kontextu } k \tag{2.21}$$

Pro kontext k_0

Znak	Počet výskytů	Pravděpodobnost výskytů
M	6	$^6/_8$
N	1	$^1/_8$
X	1	$^1/_8$

Pro kontext k_1

Znak	Počet výskytů	Pravděpodobnost výskytů
I	3	$^3/_7$
U	2	$^2/_7$
A	1	$^1/_7$
_	1	$^1/_7$

Tabulka 2.27: Pravděpodobnostní tabulky obou kontextů pro předchozí příklad

2.4.2 Adaptivnost kódování

Z pohledu adaptivnosti entropických kodérů existují tři základní kategorie: statické, semi-adaptivní, adaptivní [14, 27].

Statické

Nejprve je zapotřebí odhadnout pravděpodobnostní model na reprezentativním vzorku a uložit ho jako součást definice formátu.

Tato možnost se může projevovat značně nepřizpůsobivě a často nemusí být optimální, např. textový formát naučený na anglických slovech může být opravdu hrozný pro slova ruská nebo pro telefonní seznam.

Semiadaptivní

Tvoří ji dva průchody, první průchod zjistí aktuální rozložení znaků v řetězci. Následuje ukládání tohoto modelu do výstupního souboru. Druhým průchodem se kóduje samotný obsah vstupního řetězce.

Hlavní nevýhoda této metody spočívá v nutnosti uložení statistického modelu, který může být značně náročný na paměť. Další problém může být v několikanásobném procházení souboru (IO operace jsou poměrně časově náročné).

Adaptivní

Enkodér a dekodér vychází ze společného modelu (nebo jeho reprezentací). Po každém zakódování nebo dekódování znaku se model stejným způsobem rozšíří nebo upraví. Tímto se udržuje rozumná představa o pravděpodobnosti symbolů bez ukládání celého modelu. Základní problém je počáteční rychlost aktualizace pravděpodobnostního modelu – tedy aby co nejrychleji odpovídal skutečným informacím (a ne implicitní podobě). Tento problém se projeví hlavně u krátkých řetězců a veliké abecedy, naštěstí má mnoho chytrých způsobů řešení.

2.4.3 Aritmetické kódování

Aritmetické kódování je jedno z nejúčinnějších kódování [12, 14]. Oproti většině entropických kódování se liší ve velmi podstatné vlastnosti: Nesnaží se kódovat každý symbol zvlášť do posloupnosti celých bitů, ale celá vstupní posloupnost symbolů se převede do posloupnosti bitů. Resp. jeden znak může zakódovat do necelého počtu bitů.

Protože ideální velikost znaku ($-\log_2(p_i)$ bitů) nemusí být celé číslo, je aritmetické kódování mnohdy výrazně výkonnější.

Nepraktická implementace

Zjednodušeně lze říci, že aritmetické kódování si drží svůj výstup v podobě intervalu hodnot. Z tohoto intervalu nakonec vybere nějaké krátce reprezentovatelné číslo, které bude výstupní hodnotou, např. $\langle 0,2,0,4) \to 0,25 = [,01]b$.

Tento interval je na začátku nastaven na $\langle 0,1)$, se vstupujícími symboly se interval postupně stává menší a minimální bitová reprezentace delší.[13]

Nechť definice Σ, s_i, p_i, ... zůstávají stejné jako na začátku podkapitoly. Ke každému symbolu se přiřadí jeho interval, který je částí intervalu $\langle 0,1)$ následujícím způsobem.

$$c_i = \sum_{a=0}^{i-1} p_a \qquad \text{Je suma pravděpod. znaků } p_n; n < i. \tag{2.22}$$

$$d_i = \sum_{a=0}^{i} p_a = c_i + p_i \qquad \text{Podobně s přičtením své pravděpodobnosti.} \tag{2.23}$$

$$f(s_i) = \langle c_i, d_i) \qquad \text{Každému symbolu se přidělí interval.} \tag{2.24}$$

Další znak je převeden jako zúžení výstupního intervalu podle $f(s_i)$. Takže u velmi pravděpodobných symbolů se interval příliš nezmění, a jeho bitová reprezentace není tak omezená (tím pádem veliká).

$$\begin{aligned} vystup_{new} = zuzeni\left(vystup, f(g_x)\right) = \\ zuzeni\left(\langle a,b\rangle, \langle c,d\rangle\right) = \langle a+mc, b+md\rangle \\ m = b-a \end{aligned} \tag{2.25}$$

[13] Další přidaný znak ve skutečnosti nemusí vůbec měnit délku výstupu.

Za zmínku stojí, že když se neví, jak veliká data se kódují, je vhodné do vstupní abecedy zahrnout speciální symbol pro konec souboru. Následující algoritmy, doufám, poskytnou lepší vysvětlení:

Algoritmus 2: Aritmetické kódování pro čísla s nekonečnou přesností:

Result: číslo: číslo reprezentující výslednou posloupnost symbolů
$a \leftarrow 0$;
$b \leftarrow 1$;
for *x=1* **to** m **do** pro všechny indexy vstupních symbolů
    $aktualni\_rozsah \leftarrow b - a$;
    $i, s_i = h_x$; `/* zjištění indexu symbolu */`
    $b \leftarrow a + aktualni\_rozsah \cdot d_i$; `/* aktualizace intervalu */`
    $a \leftarrow a + aktualni\_rozsah \cdot c_i$;
`/* funkce optimum vybírá nejvhodnější číslo pro uložení z předaného intervalu */`
$cislo \leftarrow optimum(a, b)$;

Algoritmus 3: Aritmetické dekódování pro čísla s nekonečnou přesností:

Data: cislo: číslo reprezentující zakódovanou posloupnost symbolů
Result: retezec: původní data
$retezec \leftarrow \varepsilon$;
while *není konec* **do**
    $i, cisloc_i \leq cislo \wedge d_i < cislo$; `/* zjištění indexu symbolu */`
    $cislo \leftarrow (cislo - c_i)/(d_i - c_i)$; `/* aktualizace čísla */`
    $retezec \leftarrow retezec \cdot s_i$;

Praktická implementace

Pro praktickou implementaci aritmetického kodéru je nutné tento algoritmus přizpůsobit možnostem počítače [12, 14]. Problém předchozí implementace spočíval v implementaci pomocí čísel s desítkovým základem, nekonečnou přesností, nároky na paměť a čas...

Proto praktické implementace těchto algoritmů využívají celočíselnou aritmetiku s konečnou přesností, která významně zvyšuje rychlost. Dále umožňují provádět zápis výstupů z kodéru zároveň s kódováním a nezvyšují nároky na paměť s větším vstupním řetězcem.

Tento algoritmus komprese a dekomprese lze najít v [12].

2.4.4 EBCOT

EBCOT (Embedded Block Coding with Optimized Truncation) je algoritmus implementující adaptivní kontextové kódování ve formátu JPEG 2000 [24, 5, 14].

Tento algoritmus kóduje data po bitových rovinách. Nejprve kóduje nejvýznamnější bitové roviny a postupně přechází na nižší a nižší. První krok je tedy zjištění, jakou bitovou rovinu bude nejprve kódovat[14].

$$\text{první bitová rovina} = \max_{\text{číslo} \in \text{soubor}} \lfloor \log_2 (\text{abs}(\text{číslo})) \rfloor$$

[14] Je zde možnost i žádnou rovinu nezakódovat.

Zobrazení bitové roviny z původních hodnot je dáno následujícím vztahem, kde operace mod je zbytek po dělení.

$$\text{rovina}(\text{původní}, \text{index}) = \left\lfloor \frac{\text{abs}(\text{původní})}{2^{\text{index}}} \right\rfloor \bmod 2$$

Následuje kódování seřazených[15] bitů v dané rovině, nejprve v tzv. cleanup průchodu. Následuje kódování dalších bitových rovin ve třech průchodech: significance propagation, magnitude refinement a opět cleanup.

Každý vzorek je na počátku označen jako *nedůležitý*. Pokud se zakóduje první nenulový bit v nějaké bitové rovině patřící danému vzorku, tak se tento vzorek stane *důležitým*.

U příležitosti uložení prvního nenulového bitu se také ukládá i znaménko.

Cleanup průchod Kóduje ty bity, které ostatní průchody vynechaly (vysoká pravděpodobnost nuly). To znamená, že kóduje pouze *nedůležité* vzorky, jenž neměly *důležitého* souseda (alespoň na začátku tohoto průchodu). Využívá přitom 9 kontextů, které vybírá na základě *důležitosti* okolních vzorků.

Significance propagation Zakódovává vytipované vzorky, u nichž je větší pravděpodobnost jedničky (tzn. že sousedí s *důležitým*). Využívá opět 9 kontextů. Pro ztrátovou kompresi je význam tohoto průchodu významnější, protože řadí dané vzorky dopředu.

Magnitude refinement Zakódovává následující bity *důležitých* vzorků. Používá 3 kontexty.

QM-kodér Jedná se o speciální typ adaptivního stavového kontextového aritmetického kodéru, který kóduje bity ($\Sigma = \{0, 1\}$) [24, 14]. Je charakteristický především svojí orientací na bajty, tzn. že výstup tvoří N-tice bajtů. To umožňuje rychlé hledání požadované informace v souboru (část obrázku, rychlé dekódování zmenšené podoby atd.).

Aritmetické kódování bitů (na rozdíl např. od kódování bajtů) vyžaduje více volání, a tím pádem i vyšší výpočetní režii. Tento kodér to částečně řeší úpravou, u niž se nemusí násobit. Obsahuje i drobné optimalizace jako implicitně přednastavené kontexty s odhadnutými hodnotami. Zjednodušené schéma lze najít na obrázku 2.28.

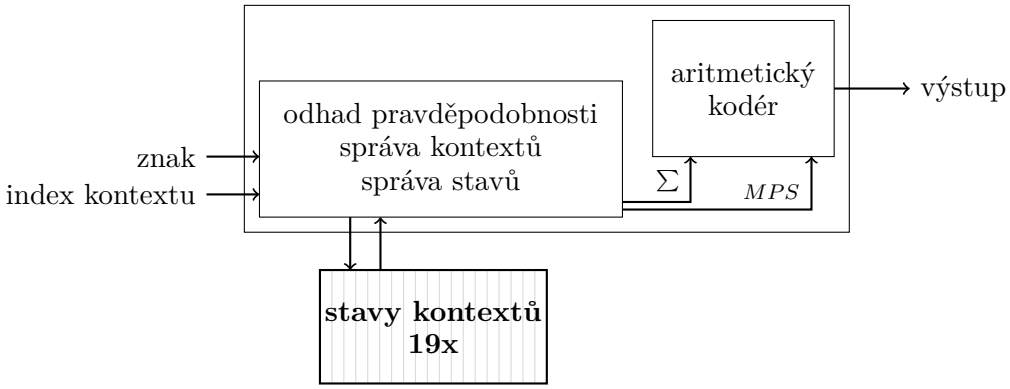
2.5 Příklady existujících řešení

Následující podkapitola pojednává o populárních formátech pro bezeztrátovou kompresi obrazu. Tyto formáty často obsahují i techniky pro ztrátovou kompresi. Vzhledem k tématu práce a komplexnosti těchto technik budou ale vynechány.

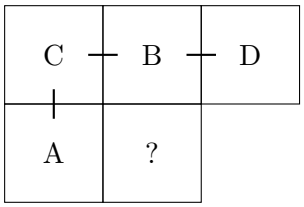
2.5.1 JPEG 2000

Tento formát podporuje ztrátovou i bezeztrátovou variantu komprese [24, 14]. V případě bezeztrátové varianty provádí barevnou transformaci pomocí RCT. Následuje N-úrovňová separabilní pyramidová dekompozice s vlnkami typu CDF 5/3 produkujících různá pásma.

[15] Jedná se o speciální seřazení.



Obrázek 2.28: Schéma QM enkodéru



Obrázek 2.29: Vzorky, ze kterých se počítá kontext ve formátu JPEG-LS; zkoumané rozdíly těchto vzorků

Tato pásma jsou ukládána ve specifickém pořadí v blocích[16] o různých velikostech (nejčastěji 64×64). Tyto bloky lze zpracovávat nezávisle, takže komplexnost kódování uvnitř nich vyváží možnost paralelní práce. Kódování se provádí algoritmem EBCOT popsaným v oddílu 2.4.4.

2.5.2 JPEG-LS

Tento formát umožňuje rychlou bezeztrátovou kompresi s vysokým kompresním poměrem [29, 14]. Díky těmto vlastnostem byl použit např. pro po posílání obrázků z Marsu na Zem[17]. Je založený na MED prediktoru (oddíl 2.3.1) a pokročilém kontextovém kódování s neuvěřitelnými 365 kontexty. Ty jsou počítané podle čtyř obklopujících hodnot (viz obrázek 2.29).

Dále je použito upravené Golombovo kódování. Toto kódování je jednou z variant kódů s proměnlivou délkou znaků a hodí se pro exponenciální rozložení hodnot. Ukázka výpočtu tohoto kódování je v příkladu 2.30.

2.5.3 JPEG XR (HD Photo)

JPEG XR profituje z PCT – reverzibilní hierarchická bloková transformace[18] příbuzná diskrétní kosinové transformaci z klasického JPEGu (lehce popsanou v oddílu 2.2.1) [15, 6, 14]. Reverzibilita je dosažena pomocí algoritmu lifting.

[16] Formát definuje celou hierarchii poskytující různé funkce – code block, tile, precinct, packet, quality layer

[17] Došlo k mírné modifikaci

[18] Pravděpodobně se jedná o podobnou transformaci jako v [17].

Příklad 2.30: Výpočet základního Golombova kódu pro jeden symbol.

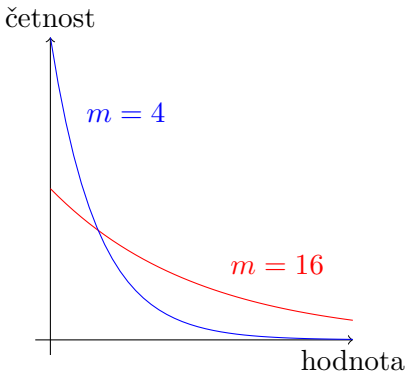
$m =$	parametr kódu vypočítaný dle předpokádaného rozložení
$c = \lfloor \log_2(m) \rfloor$	délka druhé části kódu
$n =$	kódované číslo
$q = \left\lfloor \frac{n}{m} \right\rfloor$	počet jedniček v první části kódu
$A = 11111\ldots0$	první část kódu s délkou $q+1$
$B = n - qm$	hodnota druhé části (zbytek po dělení q), délka $= c$

(a) Výpočet reprezentace jednoho znaku v Golombově kódu

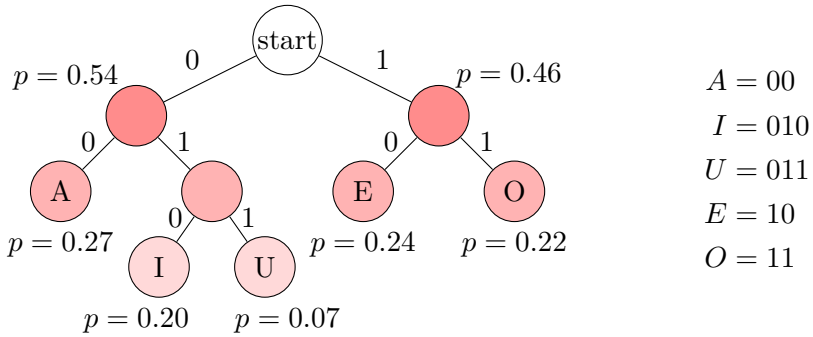
Nechť $m = 4 \Rightarrow c = 2$

$n = 0$	$\Rightarrow$ kód $=$	$0\|00$;
$n = 1$	$\Rightarrow$ kód $=$	$0\|01$;
$n = 2$	$\Rightarrow$ kód $=$	$0\|10$;
$n = 3$	$\Rightarrow$ kód $=$	$0\|11$;
$n = 4$	$\Rightarrow$ kód $=$	$10\|00$;
$n = 9$	$\Rightarrow$ kód $=$	$110\|01$;
$n = 15$	$\Rightarrow$ kód $=$	$1110\|11$;

(b) Příklad Golombova kódu



(c) Výhodnost výběru m záleží na parametrech aproximovaného geometrického rozložení



Obrázek 2.31: Ukázka kódování samohlásek české abecedy pomocí Huffmanova kódování: pravděpodobnosti samohlásek; jeden z validních pomocných stromů; výsledná podoba zakódovaných znaků (je dána průchodem ve stromu).

Obsahuje kontextové kódování s proměnlivou délkou kódu – tzv. Huffmanovo kódování. Huffmanovo kódování na rozdíl od Golombova kódu není tak pevně spjato s jedním typem rozložení pravděpodobnosti a může být i adaptivní. Ukázku je možné vidět na obrázku 2.31.

2.5.4 PNG

PNG (Portable Network Graphics) je populární bezeztrátový formát vhodný především pro kódování velkých jednobarevných ploch [22, 14]. Je založen na několika prediktorech (formát je nazývá filtry), jejich popis lze nalézt v oddílu 2.3.2.

Jako entropické kódování je použit algoritmus deflate. Deflate je odvozený od algoritmu LZ77. Tento algoritmus se řadí ke slovníkovým kompresním metodám[19].

V tomto případě existují dvě pohyblivá okna, první se vyskytuje před kódovaným symbolem a slouží jako slovník. Druhé je uvozeno právě kódovaným symbolem a reprezentuje nejdelší hledaný řetězec v prvním okně.

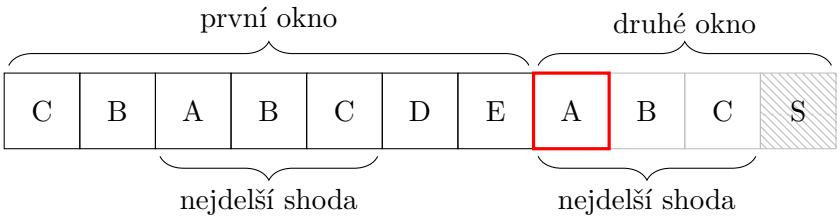
Algoritmus se snaží nahradit co možná nejdelší část druhého okna třemi hodnotami: První hodnota reprezentuje index, o kolik míst zpětně se musí dekodér vrátit, aby nalezl začátek stejného podřetězce. Druhá hodnota informuje o délce této shody. Následuje další znak po shodě.

Deflate algoritmus navíc na tyto dvojice hodnot dále použije Huffmanovo kódování (viz obrázek 2.31). Toto kódování je podobné aritmetickému, ale s tím rozdílem, že u Huffmanova kódování se symbolu přiřadí vždy celý počet bitů.

U tohoto formátu je navíc možné použít tzv. palety barev. Palety barev slouží k významné redukci barevného prostoru a uplatňují se u obrázků s malým počtem barev. V principu se např. 24 bitová RGB hodnota nahradí indexem do palety barev definované v hlavičce souboru (index má 1, 2, 4 nebo 8 bitů). Další výhodou pro jednoduché obrázky spočívá ve výrazné podpoře nižších barevných hloubek (lze použít 1, 2, 4, 8 a 16 bitů na barvu).

Všechny výše zmíněné vlastnosti umožňují tomuto formátu excelovat u jednoduchých počítačem generovaných fotek.

[19] Slovníkové metody komprese nějakým způsobem vytváří slovník (např. často kódovaných slov), indexy do tohoto slovníku mohou být použity místo výpisu vlastní hodnoty.



	reprezentace	velikost
původní kód:	ABCS	$8b \cdot 4 = 32b$
LZ77 kód:	$(5, 3, S)$	cca $3b + 2b + 8b = 13b$

Obrázek 2.32: Ukázka LZ77: Aktuálně kódovaný symbol je červeně označen

Kapitola 3

Praktická část

V této části budou shrnuty důležité výsledky, které byli získány různou kompresí vybraných obrázků pomocí vytvořeného programu. V závěru této kapitoly bude vybrán nejvhodnější postup komprese a ten bude porovnáván s aktuálně používanými formáty.

3.1 Obecné informace k výsledkům

V této části budou popsány společné prvky prezentovaných výsledků jako jsou vstupní data nebo popis tabulek, ve kterých budou výsledky zprostředkovány.

3.1.1 Datasety

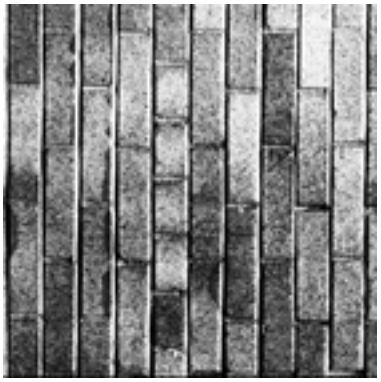
Různé kompresní metody budou zkoumány na reprezentativním vzorku obrázků. Tyto vzorky jsou roztříděny do pojmenovaných souborů obsahujících fotografie podobných vlastností. Podle průměrných výsledků lze potom odvodit vhodnost nebo nevhodnost dané metody v různém kontextu. Seznam souborů (datasetů):

Kodak Kodak Photo CD je set 24 obrázků s rozlišením 2048×3072 pixelů. Je dostupný na http://www.math.purdue.edu/~lucier/PHOTO_CD/.

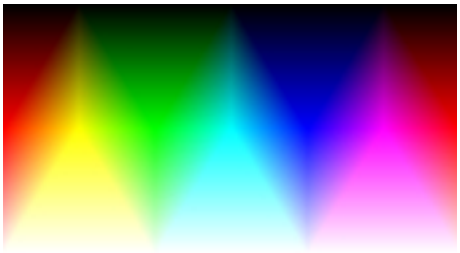
USC-misc je set 44 obrázků různých původů (16 barevných, 28 černobílých). Je dostupný na http://sipi.usc.edu/database/database.php?volume=misc.


Přírodní Jedná se o set složený z různých obrázků s přírodní tematikou pocházející z mnoha zdrojů. Některé obrázky mohou být i autorsky chráněné. Z těchto důvodů by bylo problematické zveřejňování obsahu nebo poskytování celého datasetu na CD.


Zástavba Je set složený z obrázků zachycujících člověkem vytvořené věci (nemusí se jednat pouze o domy). Obsahuje i některé odpovídající obrázky z jiných setů.




Počítačem generované Set složený z 14 obrázků zachycujících různé počítačem generované obrazy. Obsahuje pozvolné přechody, ostré hrany a velké množství jednobarevných ploch.



RGB–NIR Ukázkové RGB obrázky z datasetu RGB–NIR. Jedná se o 477 RGB obrázků z různých prostředí. Všechny obrázky jsou zmenšené o 50 % a je u nich vyvážena bílá. Podle prostředí jsou dále roztříděny do kategorií: Country, field, forest, indoor, mountain, oldbuilding, street, urban, water. Dataset je volně dostupný na http://ivrl.epfl.ch/supplementary_material/cvpr11/.

3.1.2 Tabulky

Všechna čísla vyskytující se ve výsledkových tabulkách informují o průměrném počtu bitů na jeden pixel (bits per pixel – bpp) v daném souboru (datasetu). Je proto lepší, mít tato čísla co nejnižší a přitom zajistit 100% rekonstrukci původních dat a to bez uložení dodatečných informací jinam. Proto tyto výsledky zahrnují i veškerou režii daného formátu.

Pro přehlednost jsou tabulky s výsledky dále označovány barevnými kódy, které informují o umístění daného nastavení/formátu/metody mezi ostatními v souboru (v procentech).

Nejlepší 1 %	Nejlepší 10 %	Nejlepší 20 %	Ostatní

3.2 Barevné transformace

Jedna z nejsnadněji implementovatelných komprimačních metod, která umí výrazně zlepšit výsledky komprese, je barevná transformace. Jejich vhodnost závisí na použitých barvách na snímku. Díky následující vlnkové transformaci může být více důležité, jakým způsobem reprezentují navazování barev než jednoduše pouze jejich dekorelace. Vzhledem k tomu, že tyto transformace jsou až na výjimky téměř lineární, není pravděpodobně významný vliv na následující komprimační kroky. Nejen zmíněné vlastnosti kladou další nároky na soubory obrázků, které díky tomu musí být dostatečně rozmanité.

V této práci porovnávám 7 typů barevných transformací z a do prostoru RGB: RCT (JPEG 2000) [24], YCoCg-R [23], RDgDb [23], LDgEb [23], mRDgDb [23], mLDgEb [23], LOCO-I [29]. Konkrétní definice lze nalézt v příloze A.1.

Následující tabulky porovnávají konkrétní barevné transformace, je zde použita pyramidová transformace s vlnkou 5/3 (do maximální úrovně) následuje EBCOT (jde vlastně o aproximaci JPEGu 2000).

Barevný prostor	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
RGB	10,087	13,609	11,003	10,622	2,9797
RCT	7,6670	13,588	9,4564	9,2449	2,5716
YCoCg-R	7,6323	13,544	9,4416	9,3633	**2,5028**
RDgDb	7,6486	**13,486**	9,5014	9,5325	2,6260
LDgEb	**7,5719**	13,526	**9,3949**	9,3870	2,6967
mRDgDb	7,6603	13,691	9,5491	9,5604	2,6489
mLDgEb	**7,5811**	14,191	9,4630	9,3842	2,7429
LOCO-I	7,6881	13,903	9,6122	**9,2008**	2,5311

Tabulka 3.7: Porovnání barevných transformací

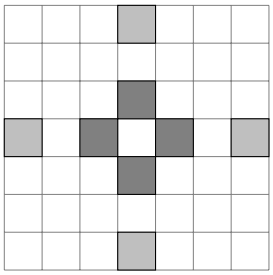
Barevný prostor	Dataset (bpp)					
	Country	Indoor	Oldbuilding	Urban	Water	Celkově
RGB	12,458	10,904	11,068	11,749	11,710	12,387
RCT	**11,686**	**10,551**	**10,231**	**10,884**	**11,073**	**11,554**
YCoCg-R	11,700	10,576	10,270	10,894	11,082	**11,559**
RDgDb	12,110	11,362	10,720	11,372	11,721	12,046
LDgEb	11,924	10,903	10,532	11,142	11,392	11,800
mRDgDb	12,112	11,366	10,722	11,375	11,723	12,048
mLDgEb	11,907	10,868	10,518	11,128	11,363	11,779
LOCO-I	11,843	10,823	10,381	11,050	11,256	11,729

Tabulka 3.8: Výkon různých barevných transformací v setu RGB–NIR a jeho vybraných součástí.

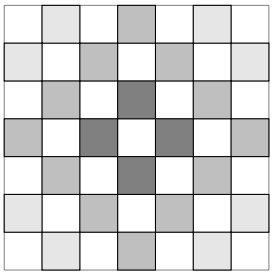
3.2.1 Výsledky barevných transformací

Běžné obrázky Ukázalo se, že RCT a YCoCg-R mají dobrý a vyrovnaný výkon napříč všemi datasety. Zajímavé bylo, že v datasetu RGB–NIR byly v podstatě bezkonkurenční (tabulka 3.8), kdežto u setů v první tabulce je vidět mnohem vyšší konkurenceschopnost ostatních barevných transformací (zejména v setu kodak). Pravděpodobně za tímto jevem stojí overfitting na podobných, ne-li stejných, datasetech.

Vzhledem k předchozím informacím a výsledkům hlavně v setu RGB–NIR lze považovat za více vhodnou RCT, i když rozdíl oproti YCoCg-R je opravdu zanedbatelný.



(a) Vstupy zařazené do výpočtu u delší vlnky v červeno-černé vlnkové transformace.



(b) Ideální okolí zařazené do výpočtu u červeno-černé vlnkové transformace.

Obrázek 3.9: Rozdíl v pojetí výpočtu červeno-černé vlnkové transformace

Počítačem generované obrázky U počítačem generovaných obrázků je obecně výkon barevných transformací velmi závislý na zvolených obrázcích a tak nelze brát výsledky nijak kategoricky.

Kompresní poměry napříč barevnými transformacemi jsou relativně velmi nevyrovnané, nicméně se ukazuje, že `YCoCg-R`, `LOCO-I` nebo `RCT` může být vhodnou volbou.

3.3 Dekompozice obrazu

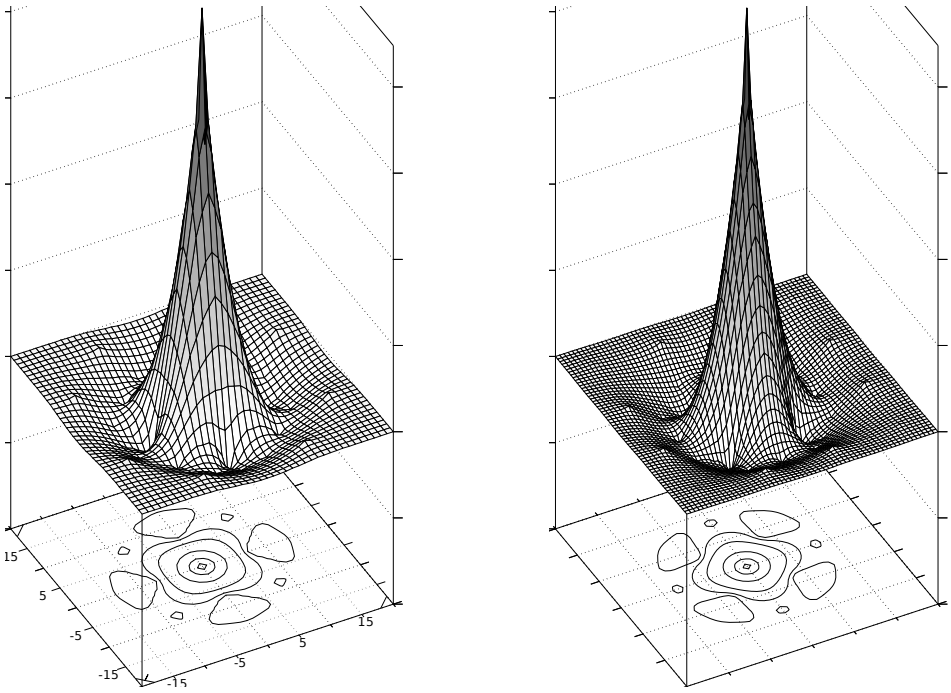
Výběr metody dekompozice hraje důležitou roli při zpracování obrázků. Bohužel jejich prakticky nekonečný počet značně ztěžuje kompletní průzkum. Proto tato práce obsahuje výběr těch nejčastěji používaných pro zpracovávání obrázků.

Podle principu chování lze odlišit dvě základní dekompozice: Separabilní a červeno-černou. Separabilní dekompozice transformuje data zvlášť, nejprve po řádcích a následně po sloupcích. V této práci je zmínka o pyramidové dekompozici, ta patří do třídy separabilních. U červeno-černé dekompozici je směr výpočtu nedůležitý, protože zachycuje všesměrové hrany.

3.3.1 Odlišnost červeno-černé vlnkové transformace

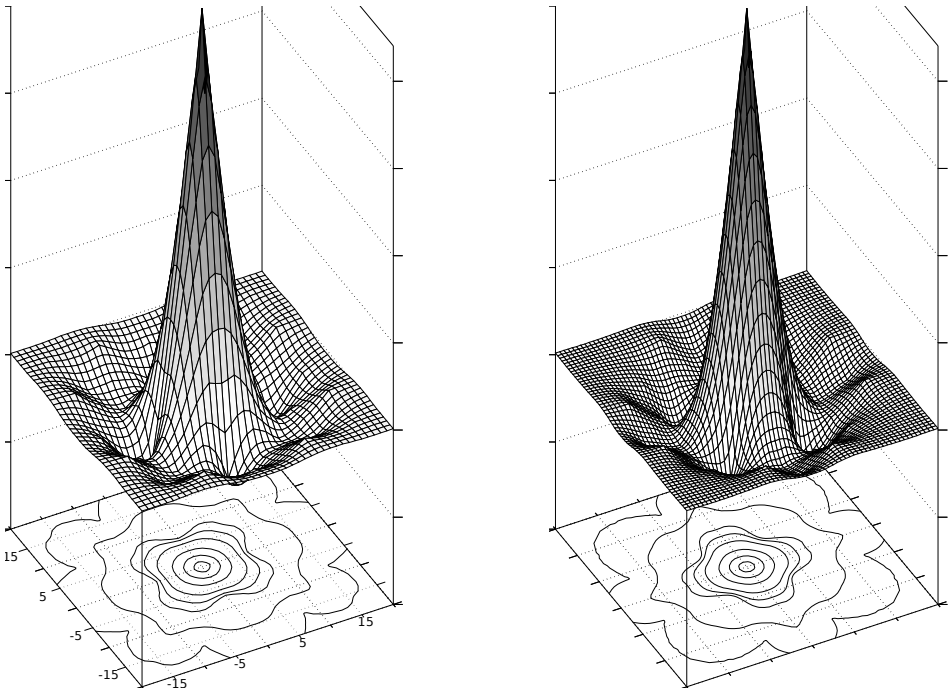
Protože zdroj přesně neinformuje o korektní implementaci delších vlnek (resp. o úpravě liftingových kroků, jejichž vstupem nejsou pouze přímí sousedé zpracovávaného vstupu) bylo nutné provést jisté úpravy i pro tyto případy.

U krátkých vlnek se tedy nejedná o změnu oproti zdrojové implementaci. U dlouhých vlnek byly zahrnuty pouze 4 směry ve výpočtu a ostatní jsou ignorovány (obrázek 3.9). Tento způsob byl zvolen, protože 2D filtry by pravděpodobně nebylo možné prakticky užít kvůli jejich výpočetní náročnosti. Implementace pouhých 4 směrů umožňuje snížit náročnost na srovnatelnou úroveň, jako je tomu u pyramidové dekompozice při současném minimálním omezení výběru vlnek. Grafy bázových funkcí krátkých a dlouhých liftingových kroků je možné porovnat na obrázku 3.10.



(a) 5/3 – černé pásmo (vysoké frekvence)

(b) 5/3 – žluté pásmo (střední frekvence)



(c) 9/7-M – černé pásmo (vysoké frekvence)

(d) 9/7-M – žluté pásmo (střední frekvence)

Obrázek 3.10: Rozdíly dlouhých (9/7-M) a krátkých (5/3) vlnek u zkoumané implementace červeno-černé vlnkové transformace.

3.3.2 Značení dekompozice obrazu

Aby bylo možné se o těchto transformacích efektivně zmiňovat, zavádím zjednodušení popisu následujícím způsobem:

`T[a,b]` Separabilní dekompozice s maximálním počtem úrovní (rozklad na 2 pásma) nastavených na hodnotu `a`. Vedlejší pásma jsou dále rozkládána `b`-úrovňovou uniformní dekompozicí. Toto chování je ilustrováno na obrázcích 3.11.

`Pyr[a]` Klasická pyramidová dekompozice do úrovně `a`. Stejné jako `T[2a,1]`.

`Uni[a]` Uniformní separabilní dekompozice do úrovně `a`. Všechna pásma mají přibližně stejnou velikost. Shodné s `T[a,a-1]`.

`Rb[a,b]` Podobně jako `T[a,b]` s rozdílem, že se používá červeno-černá vlnková transformace.

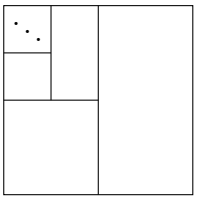
3.3.3 Výběr druhu dekompozice obrazu

EBCOT byl se svými kontexty uzpůsoben pro pyramidovou dekompozici, protože se kontexty navzájem překrývají, není penalizace ostatních dekompozic tak výrazná. I tak je otázkou, jaký vliv bude mít přizpůsobení entropického kodéru dané dekompozici.

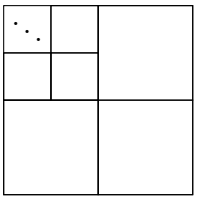
Tabulky Následující tabulka informuje o vlastnostech různých dekompozic pro různé kodéry. U barevných obrázků byla nejprve provedena RCT, potom daná dekompozice s vlnkou 5/3. Pro entropické kódování byl použit EBCOT, u druhé z těchto tabulek je použito dříve popsané aritmetické kódování.

Dekompozice	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
T[∞,0]	**7,5752**	**6,3871**	**9,3387**	**8,7171**	**1,7286**
T[∞,1]	7,6670	6,4118	9,4176	8,7726	1,8911
T[∞,2]	7,9422	6,5231	9,6692	9,0069	2,2538
T[∞,3]	8,1911	6,6202	9,9164	9,1931	2,6533
T[∞,4]	8,4389	6,7632	10,153	9,3931	3,0777
Rb[∞,0]	7,9334	6,5918	9,6583	9,0727	2,2118
Rb[∞,1]	8,0474	6,6318	9,7503	9,1647	2,4206
Rb[∞,2]	8,3889	6,8236	10,084	9,4636	3,0570
Rb[∞,3]	8,5624	6,8904	10,234	9,5969	3,3841
U[0]	10,379	7,3178	11,876	11,091	3,2447
U[1]	8,8747	6,9181	10,565	9,8173	2,7132
U[2]	8,0261	6,6122	9,7346	9,0994	2,2999
U[3]	8,0729	6,5839	9,7704	9,1127	2,3282
U[4]	8,1613	6,5667	9,8830	9,1505	2,4773
U[5]	8,3640	6,6441	10,074	9,3149	2,7926
U[6]	8,5449	6,7176	10,227	9,4527	3,1594

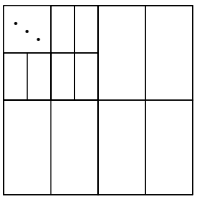
Tabulka 3.12: Porovnání různých dekompozic při entropickém kódování EBCOTem.



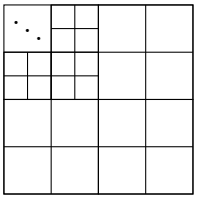
T[∞,0]



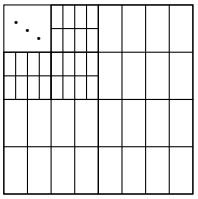
Pyr[∞], T[∞,1]



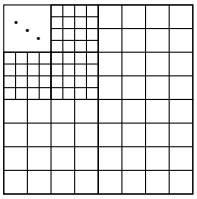
T[∞,2]



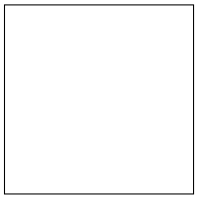
T[∞,3]



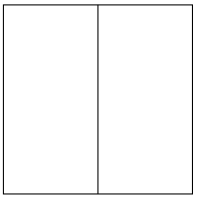
T[∞,4]



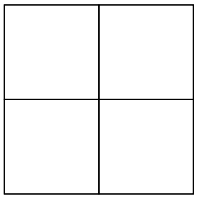
T[∞,5]



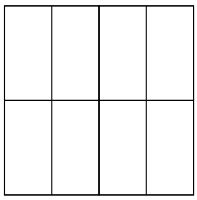
U[0], P[0], T[0,X]



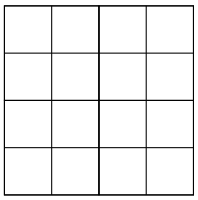
U[1], T[1,X]



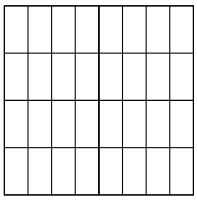
U[2], T[2,1]



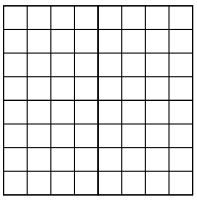
U[3], T[3,2]



U[4], T[4,3]



U[5], T[5,4]



U[6], T[6,5]

Obrázky 3.11: Vizualizace různých separabilních dekompozic s příslušným označením.

Dekompozice	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
T[∞,0]	**8,7957**	6,7312	**10,375**	9,8602	**2,9315**
T[∞,1]	**8,7773**	**6,6898**	**10,391**	**9,7512**	3,0121
T[∞,2]	8,9640	6,7905	10,561	9,9033	3,5015
T[∞,3]	9,1300	6,8575	10,761	10,027	3,8794
T[∞,4]	9,3503	6,9962	10,978	10,225	4,4037
Rb[∞,0]	8,9622	6,9266	10,506	10,063	3,7136
Rb[∞,1]	9,0424	6,9912	10,578	10,109	4,1410
Rb[∞,2]	9,2154	7,0674	10,767	10,220	4,7160
Rb[∞,3]	9,3902	7,1703	10,933	10,370	5,2865
U[2]	12,310	8,1591	13,479	12,593	6,1081
U[3]	10,913	7,6559	12,237	11,458	4,9001
U[4]	10,050	7,2567	11,536	10,723	4,3499
U[6]	9,6394	7,0799	11,235	10,434	4,5643
U[8]	9,8004	7,2260	11,442	10,594	5,3511
U[10]	10,124	7,6372	11,739	10,884	6,2439

Tabulka 3.13: Porovnání různých dekompozic při aritmetickém kódování hodnot.

Separabilní Jako nejlepší dekompozice se jednoznačně ukázala `T[X,0]`. Důvodem k tomuto výsledku může být rozšiřování oboru hodnot a pravděpodobné snížení jejich korelace při každém dalším průchodu WT. Výhodou pyramidové dekompozice (`T[X,1]`) nad `T[X,0]` je symetričnost a pravděpodobně i lepší rozlišení relevance informací, které je relevantní pro ztrátovou kompresi. To by mohl být zřejmě důvod, proč je upřednostňována v JEPG 2000. Nevýhodou je vyšší režie o cca 30 % [14].

U aritmetického kódování se výsledky přiklání k pyramidové dekompozici (`T[X,1]`) proto, že v H pásmu se vyskytují vertikální pruhy stejných hodnot. Další rozkládání vedlejších pásem zde překvapivě kazí kompresní poměr. Tento jev lze přičíst dekorelaci hodnot (žádné pruhy v těchto směrech už nejsou), takže další průchod pouze rozšiřuje obor hodnot. Každopádně by mohlo být zajímavé provést další průchod v horizontálním (pro LH), ve vertikálním (pro HL) a diagonálním (pro HH) směru, kde lze ještě očekávat jistou závislost.

Z předchozích důvodů bych dále rozebíral hlavně `T[X,0]`, protože jde o lepší a rychlejší variantu.

Červeno-černé Červeno-černá dekompozice (`Rb`) podobně profituje z minimální transformace vedlejších pásem. Tentokrát není tak patrný rozdíl v chování EBCOTu a aritmetického kódování.

Při kódování EBCOTem je vhodnější zvolit stejně počítaný kontext jako LL pásmo, a to u všech podpásem. Pravděpodobně zde hraje vliv její nezávislost na směru hran, a tím pádem kódování jakýchsi obrysů objektů původního obrázku. V tomhle ohledu by mohlo být zajímavé použít prediktor nebo jiný způsob využití tohoto chování.

Tato dekompozice není nejlepší ani při jednom druhu kódování, avšak je důležité myslet na to, že lze očekávat více vlivu použité vlnky. Z rozdílu v `Rb[X,0]` a `Rb[X,1]`, v porovnání s rozdílem `T[X,0]` a `T[X,1]` vzhledem k implementaci, lze usoudit, že EBCOT není tak

vhodný pro `Rb[X,0]`. Zkoumaná implementace totiž černé pásmo rozdělí na dvě pásma i když neproběhla černá obměna modro-žlutého liftingu. Otázkou je, jestli potřebná komplexnost ve zpracování šachovitého pásma speciálním algoritmem byla prospěšnější než separabilní dekompozice.

U aritmetického kódování se pravděpodobně kontraproduktivně zvyšuje počet výsledných vzorku počítající s hranami (a tak zvyšuje čísla vzdálená od nuly).

Uniformní Uniformní dekompozice nebyly pro tento účel příliš vhodné, dokonce lze říct, že byly nejhorší. Zajímavé vlastnosti se ukázaly pouze při aritmetickém kódování. Zde došlo s dalším rozkladem vedlejších pásem ke zlepšení (na rozdíl od `T[X,Y]`, kde bylo zhoršení). Lze to ale připsat souběžnému zmenšování LL pásma.

3.3.4 Výsledky `T[X,0]` podle počtu úrovní

Existuje zde možnost, že provádění vlnkové transformace by mohlo být s některým dosaženým pásmem nevýhodné. Například by mohl první průchod vlnkové transformace odstranit všechen šum a výsledek by potom mohl být beze ztráty na kompresním poměru kódován jako LL pásmo EBCOTem.

Tím by se mohlo ukázat, že další rozkládání je zbytečné nebo dokonce by velmi malá pásma neposkytovala tak ucelený kontext a rozkládání by znevýhodňovala. Vzhledem k velikosti hlaviček v použitém formátu nelze předpokládat, že by zde byl v tomto ohledu nějaký výrazný vliv (možná na úrovni zaokrouhlení čísel).

Tabulka U následující tabulky byla použita RCT, `T[?,0]` s vlnkou 5/3, EBCOT.

`Počet úrovní`	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
4	7,6598	6,4299	9,4162	8,7756	1,8540
6	7,5940	6,3922	9,3569	8,7288	1,7585
8	7,5793	6,3853	9,3427	8,7193	1,7334
10	**7,5758**	**6,3848**	9,3393	**8,7173**	**1,7282**
12	**7,5751**	6,3853	**9,3385**	**8,7169**	**1,7276**
14	**7,5750**	6,3860	**9,3385**	**8,7169**	**1,7279**
16	**7,5750**	6,3866	**9,3385**	**8,7169**	**1,7282**

Tabulka 3.14: Porovnání podle počtu provedených úrovní dekompozice

Výsledky Výsledky prozrazují, že nejvýhodnější bude použít kolem 11 úrovní.

Za povšimnutí stojí, že v datasetu USC-misc je množství malých obrázků a i ideální výsledek je relativně posunut. Proto se dá očekávat, že by mohla být lepší závislost podle velikosti posledního LL pásma než podle počtu úrovní, které na velikosti více či méně nehledí.

3.3.5 Výsledky podle minimální velikosti LL (`T[X,0]`)

Následující pokus zjišťuje, jestli je vhodné se rozhodovat na základě velikosti zbylého pásma.

Tabulka Bylo použito stejné nastavení komprese jako v předchozím pokusu (RCT, 5/3, EBCOT). Vychází ze zjištění v předchozím pokusu.

`Minimální velikost pro rozklad`	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
`257`	7,5850	6,6571	9,3564	8,7289	1,9481
`129`	7,5772	6,4401	9,3421	8,7193	1,7994
`65`	7,5753	6,3935	9,3388	8,7173	1,7461
`33`	**7,5750**	**6,3854**	**9,3384**	**8,7169**	1,7317
`25`	**7,5750**	**6,3846**	**9,3384**	**8,7169**	**1,7284**
`21`	**7,5750**	**6,3846**	**9,3384**	**8,7169**	**1,7285**
`17`	**7,5750**	**6,3846**	**9,3384**	**8,7169**	**1,7284**
`13`	**7,5750**	**6,3851**	**9,3385**	**8,7169**	**1,7279**
`9`	**7,5751**	**6,3851**	**9,3385**	**8,7169**	**1,7280**
`5`	7,5751	**6,3857**	9,3386	**8,7170**	**1,7281**
`3`	7,5752	**6,3864**	9,3387	8,7170	**1,7283**
`2`	7,5752	**6,3871**	9,3387	8,7171	**1,7286**

Tabulka 3.15: Porovnání podle povolené minimální velikosti LL pásma

Výsledky Na rozdíl od předchozího rozdělení podle počtu úrovní se tyto výsledky zdají být nejen lepší, ale hlavně i stabilnější.

Tabulka ukazuje, že nejvhodnější je zastavit rozkládání u pásem velikosti 16 a méně. Pravděpodobně zde hraje vliv omezení ve výpočtu kontextu, a to:

a) Vytrénovat 19 kontextů na 256 hodnotách může být neefektivní

b) Podíl okrajových pixelů činí min. 30 %

3.4 Vlnky

Z práce [9, 32, 14] byly vybrány liftingové implementace vlnek, které by mohly být vhodné pro bezeztrátovou kompresi.

Vzhledem k charakteru červeno-černé vlnkové transformace nebylo možné implementovat vlnky, jež nějakým způsobem porušovaly definici líné vlnkové transformace (SPB, SPC).

Jde konkrétně o 5/3[14], 9/7-F[9] (4 kroková), 9/7-M[9] (2 kroková), 5/11-C[9], 5/11-A[9], 13/7-F[9], 13/7-C[9], 6/14[9], 2/6[32], 2/10[32], 1/3(Lineární)[14], Haarova[14].

Dále byly vytvořeny další vlnky z jejich podobně pojmenovaných předloh 5/3*, 1/3*, 9/7-M*, 1/7, 1/7*.

3.4.1 Výsledky pro separabilní transformace

Vzhledem k tomu, že formát JPEG 2000 používá speciální vlnku (5/3) pro bezeztrátovou kompresi, tak by odlišný výsledek byl překvapující. I přesto je nutné ověřit různé typy, protože `T[X,0]` se přeci jenom trochu liší od `T[X,1]`.

Tabulka Následující test je vytvořen s RCT, `T[X,0]` a EBCOTem. Kompletní výsledky lze dohledat v tabulce B.1.

`Vlnka`	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
`5/3`	7,5752	**6,3871**	9,3387	8,7171	1,7286
`5/3*`	**7,4848**	**6,3867**	**9,2605**	**8,6761**	1,7153
`1/3*`	**7,4929**	6,4607	9,3617	8,6905	**1,5505**
`9/7-M*`	7,6363	6,4053	9,3850	8,6877	1,8619
`5/11-A`	7,6824	6,4380	9,4142	8,8047	2,0311
`1/7*`	7,7615	6,5550	9,6122	8,8231	1,7974
`Haarova`	8,5082	6,5198	10,155	9,5265	1,5896

Tabulka 3.16: Vliv různých vlnek na výkon komprese (rozklad `T[X,0]`)

Běžné obrázky Z testování jak u klasické pyramidové dekompozice (`T[X,1]`) tak i `T[X,0]` vyšlo najevo, že vlnky typu `5/3` jsou jednoznačně nejlepší. Zajímavé vlastnosti mají i úpravy této vlnky, i přes zdánlivě marginální přičtení jedničky pro zaokrouhlení se výsledek velmi dramaticky lišil.

Dominance `5/3` lze vysvětlit mnoha způsoby. Je to relativně krátká vlnka, takže tolik nepočítá s okolním šumem nebo přechody objektů, a přitom si zachovává překvapivě dobrou predikci. Více nulových momentů se zdá být zbytečné asi proto, že v obrázcích bývají buď ostré přechody nebo velmi pozvolná změna barvy.

Počítačem generované obrázky Výborný výsledek pro počítačem generované obrázky zaznamenala lineární (`1/3`) a Haarova transformace. Tento výsledek je pochopitelný: Detaily se počítají jako lineární aproximace dvou vedlejších bodů (stejně jako u `5/3`). Vzhledem k tomu, že u počítačem generovaných obrázků pravděpodobně nebude šum takový problém, nemusí se už uměle generovat v update kroku (např. zaokrouhlováním), a proto žádný není.

3.4.2 Výsledky pro červeno-černou dekompozici

Výběr vlnky by mohl u červeno-černé dekompozice hrát větší vliv, než je tomu u separabilních. Z tohoto důvodu je důležité ověřit, jestli se změnou vlnky nedojde k výraznému skoku.

Tabulka Následující test byl vytvořen s RCT, červeno-černou dekompozicí a EBCOTem. Kompletní výsledky lze dohledat v tabulce B.2.

Vlnka	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
1/3	7,8806	6,6159	9,6526	9,0212	2,1718
1/3*	**7,7815**	**6,6028**	**9,5654**	**8,9614**	**2,0415**
9/7-M*	7,8698	6,6216	9,6033	8,9722	2,4622
5/3*	7,8757	6,6139	9,5918	9,0452	2,2799
1/7*	7,8774	6,6739	9,6626	8,9941	2,3686
5/11-C	8,0905	6,6835	9,7887	9,1863	2,8255
13/7-T	8,1863	6,6753	9,8485	9,2074	3,0613

Tabulka 3.17: Vliv různých vlnkek na výkon komprese (červeno-černá WT)

Výsledky K výraznému skoku nedošlo, a proto pořád platí předchozí zjištění. Kdyby se podařilo nějakým způsobem zlepšit její výkon, mohla by být výhodná vlnka 1/3*, 9/7-M* nebo 5/3*.

3.5 Prediktory

Prediktory lze v transformacích použít mnoha způsoby. Důležité je si uvědomit, že prediktory nejsou lineární operace, a tak by mohlo hrát roli i v jaké fázi komprese je použít. Je implementováno 6 různých prediktorů. Jedná se o left [22], up [22], diagonal, MED [11], Paeth [22], GAP [11, 31]. Prediktory left, up, diagonal používají predikci podle hodnoty vlevo, nahoře nebo nahoře vlevo. Definice ostatních prediktorů je možné vyčíst v podkapitole 2.3.

3.5.1 Výsledky použití prediktoru před vlnkovou transformací

Jednou z možností, jak použít prediktor, je predikovat hodnoty před WT. Tato predikce by mohla snížit množství ukládaných detailů za předpokladu, že prediktor bude-li dostatečně přesný nebo budou-li chyby vhodně korelované.

Prediktor	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
Žádný	**7,5752**	**6,3871**	**9,3387**	**8,7171**	1,7286
Left	8,5090	6,6658	10,211	9,3882	**1,6952**
Up	8,2142	6,6327	10,051	9,2060	1,7724
Diag	8,8802	6,9294	10,633	9,8203	1,9540
MED	8,5151	6,6278	10,263	9,3103	**1,6921**
Paeth	8,6080	6,6917	10,305	9,4326	1,8074
GAP	12,622	8,5783	13,957	12,575	3,2783

Tabulka 3.18: Vliv použití různých prediktorů na původní pásmo před vlnkovou transformací

Běžné obrázky Prakticky se zdá, že tato strategie nefunguje. Prediktor pravděpodobně koliduje s WT tím, že vytváří nesouvisející skoky v datech. Tyto skoky mohou být hůře predikovatelné, než je tomu u původních hran v obraze, a tak nejsou dobře komprimovatelné.

Počítačem generované obrázky Jediná aplikace této strategie může být u počítačem generovaných obrázků. V tomto ohledu je nutné vyzkoušet některé vlnky, které se ukázaly být vhodné na tomto typu obrázků.

Prediktor	Dataset (bpp)
	Poč. gen.
Žádný	**1,5573**
Left	1,6519
Up	1,7066
Diag	1,8356
MED	1,6840
Paeth	1,6991
GAP	3,0206

(a) Lineární transformace

Prediktor	Dataset (bpp)
	Poč. gen.
Žádný	1,5896
Left	**1,5693**
Up	1,6936
Diag	1,8356
MED	1,6285
Paeth	1,7282
GAP	2,9056

(b) Haarova transformace

Tabulky 3.19: Výsledky použití prediktoru před WT pro počítačem generované obrázky.

I v tomto směru se nevyplatí používat prediktor. Pouze Haarova transformace mírně profituje, ale lineární se zdá být lepší.

3.5.2 Výsledky použití prediktoru před EBCOTem

Další způsob je použít prediktor na různá kódovaná pásma, např. ještě před prací EBCOTu. V případě vhodné predikce by to mohlo snížit entropii dat. Například LH pásmu se vyskytují horizontální hrany, které lze vynulovat `Left` prediktorem.

Následující tabulka ukazuje LH pásmo kódované jako LL (stejně).

Prediktor	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
Žádný	**7,5752**	**6,3872**	**9,3387**	**8,7171**	**1,7287**
Left	7,8499	6,5249	9,6349	8,9857	1,8191
Up	7,9089	6,5512	9,6908	9,0601	1,8951
Diag	8,0622	6,6393	9,8331	9,2020	1,9439
MED	7,9079	6,5377	9,6915	9,0434	1,9062
Paeth	7,9717	6,5896	9,7447	9,1211	1,8862
GAP	7,7825	6,5268	9,5748	8,9431	1,9114

Tabulka 3.20: Použití prediktoru na LH pásmo před kódováním EBCOTem

V tomto směru se ovšem ukázalo, že tato predikce vhodně nespolupracuje s následujícím EBCOTem. Ten již kontext využívá a prediktor tomu pravděpodobně ztěžuje. Dokonce i při základním aritmetickém kódování prediktory zhoršovali kompresní poměr (včetně

červeno-černé WT). Potom je na vině nevhodná predikce, která si neumí poradit s výstupy po WT.

3.5.3 Výsledky použití prediktoru místo další úrovně WT

V předchozím pokusu bylo zjištěno, že nemá smysl rozkládat pásma až do velikosti jednoho pixelu. Zajímavé by proto mohlo být zjištění, jaký vliv mohou mít prediktory na této hranici, popřípadě jestli nebude použití prediktoru na LL pásmo v nějakém bodě výhodnější.

Závislost na sudých a lichých úrovních V pokusech se objevil nečekaný vliv `T[X,0]`. Tato dekompozice umožňuje mít pásmo obdélníkové (1, 3, ... úroveň) a čtvercové (0, 2, ... úroveň). Podle toho byl i rozdíl ve výkonech různých prediktorů.

Prediktor	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
Žádný	7,7920	6,5025	9,5287	8,8901	2,0088
Left	7,6699	6,4241	9,4309	8,8222	1,6490
Up	**7,5980**	**6,3792**	**9,3734**	**8,7414**	**1,5987**
Diag	7,7177	6,4460	9,4757	8,8628	1,6827
MED	7,6066	**6,3778**	9,3841	8,7476	1,6087
Paeth	7,6390	6,4087	9,4124	8,7887	1,6377
GAP	7,9352	6,5824	9,6708	9,0142	2,0372

(a) Tři úrovně WT

Prediktor	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
Žádný	7,6598	6,4300	9,4162	8,7756	1,8540
Left	7,5976	6,3861	9,3647	8,7420	1,6693
Up	7,6051	6,3858	9,3754	8,7401	1,6722
Diag	7,6371	6,4077	9,4003	8,7790	1,7100
MED	**7,5895**	**6,3743**	**9,3615**	**8,7276**	**1,6605**
Paeth	7,6031	6,3894	9,3733	8,7451	1,6802
GAP	7,7382	6,4713	9,4927	8,8470	1,8699

(b) Čtyři úrovně WT

Tabulky 3.21: Různé výsledky podle úrovně v jaké byl prediktor použit

U sudých LL pásem (0,2,...) je výhodné použít prediktor `MED` (potom `left` a `Paeth`). U lichých LL pásem (1,3,...) je vhodnější použít prediktor `up` (potom `MED` a `Paeth`). Důvod bude zřejmě to, že po horizontálním separabilním průchodu vypadá L pásmo, vzhledem k původnímu, roztáhnutě, a tím pádem je vhodnější se spoléhat na prediktor `up`. Jinak platí, že prediktor `MED` je obecně nejvýhodnější.

Výsledky pro různé úrovně Nejprve je vhodné zjistit, jestli je vhodnější prediktor nebo vlnková transformace. V případě, že je vhodnější WT, je dobré vědět, jestli to platí na všech úrovních.

V následující tabulce je používán prediktor na aproximaci před uložením pomocí EBCOTu (vlnka 5/3*). U sudých úrovní byl použit prediktor `MED`, u lichých `Up`.

Úrovní	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
0	7,8855	**6,3503**	9,7091	8,8910	**1,1190**
1	7,5689	6,3950	9,4546	8,7700	1,3295
2	7,5128	6,3761	9,3233	8,7186	1,5100
3	7,5097	6,3788	9,2969	8,7009	1,5826
4	7,4986	6,3738	9,2829	8,6864	1,6455
5	7,4902	6,3781	9,2703	8,6800	1,6738
6	**7,4850**	6,3783	**9,2638**	**8,6761**	1,6919
8	**7,4839**	6,3818	**9,2603**	**8,6751**	1,7062
10	**7,4841**	6,3838	**9,2599**	**8,6754**	1,7115

Tabulka 3.22: Výsledky pro používání příslušných prediktorů na různé úrovně

Výsledky nejsou překvapující: V setu kodak, přírodním a zástavbě je vhodnější vlnková transformace. U počítačem generovaných obrázků se vyplatí WT vůbec nepoužívat a raději se spolehnout na `MED` prediktor a ihned použít EBCOT. Toto chování je zřejmě dáno charakterem změny barev v daných setech.

V setu USC-misc jsou výsledky velmi vyrovnané, zdá se, že tento set není vhodný jak pro WT, tak pro prediktor. Bude to pravděpodobně velkou různorodostí původu fotografií, které mimo jiné obsahuje směs obou předtím zmíněných typů obrázků.

Výsledky pro velikost LL pásma Předchozí test neumožnil kvalitně zjistit, jestli prediktory umožňují zlepšení kompresního poměru i u běžných obrázků. Chybí především informace týkající se posledních úrovních WT.

Následující tabulka porovnává dvě možnosti komprese. První ponechává vše na vlnkové transformaci a prediktor nepoužívá, druhá před EBCOTem použije `MED` prediktor. Obě zastavují rozkládání v různých velikostech LL.

Vel.	Dataset (bpp)		
	Kodak	Přírodní	Zástavba
1025	7,5119	9,3033	8,6861
513	7,4904	9,2751	8,6750
257	**7,4843**	9,2619	**8,6741**
129	**7,4840**	**9,2599**	8,6750
65	**7,4842**	**9,2597**	8,6755
33	7,4844	**9,2600**	8,6757
17	7,4845	**9,2601**	8,6759
9	7,4846	9,2602	8,6759
5	7,4847	9,2603	8,6760

(a) S MED prediktorem

Vel.	Dataset (bpp)		
	Kodak	Přírodní	Zástavba
1025	7,6864	9,4802	8,8787
513	7,5298	9,3375	8,7234
257	7,4944	9,2780	8,6879
129	7,4867	9,2638	8,6783
65	**7,4849**	**9,2605**	**8,6763**
33	**7,4846**	**9,2601**	**8,6759**
17	**7,4846**	**9,2602**	**8,6759**
9	**7,4846**	**9,2602**	**8,6759**
5	**7,4847**	**9,2603**	**8,6760**

(b) Bez prediktoru

Tabulky 3.23: Vliv prediktoru na vhodnost ukončení vlnkové transformace při různých velikostech LL pásma

Prediktory umožňují dřívější ukončení WT (menší počet úrovní) s relativně minimálním ziskem v kompresním poměru. Svoje uplatnění může tato strategie mít u malých obrázků. Prakticky je otázkou, jestli tato zlepšení kompresního poměru nevyváží zvýšení komplexnosti komprimačního řetězce. Z těchto důvodů se používání prediktorů nezdá výhodné pro běžné obrázky.

3.6 Celkové výsledky

Následující podkapitola se bude věnovat porovnání veřejných a často používaných formátů a několika navrhovaných řešení, která logicky vyústila z předchozích výsledků v praktické části.

3.6.1 Zkrácené představení formátů

Pro porovnání výsledků byly připraveny čtyři uznávané standardy umožňující bezeztrátovou kompresi: JPEG 2000, JPEG XR, JPEG LS a PNG. Tyto formáty jsem srovnal s různým nastavení knihovny WTlib, která je výsledkem této práce. Popis těchto formátu lze nalézt v podkapitole 2.5.

V této části je projednáván pouze způsob získání souborů, se kterými se později porovnávají výsledky. I když se jedná o určité standardy, a proto lze očekávat jakousi jednoznačnost zobrazení předlohy, opak je mnohdy pravdou. Prakticky jde pouze o souhrn povinných a použitelných technik, které musí zvládat každý dekodér. O vhodnosti a způsobu použití může být u enkodérů výrazný rozdíl.

U bezeztrátové komprese naštěstí nejsou rozdíly tak razantní jako u komprese ztrátové. Je to dáno tím, že algoritmus rozhodnutí o důležitosti každého bitu není často plně definovaný příslušným formátem, a tak jej bez porušení standardu mohou programy různě úspěšně optimalizovat[1]. Proto může hrát roli i to, jak byl některý obrázek získán.

JPEG 2000 Jedná se o bezeztrátovou podobu formátu JPEG 2000 prováděnou programem convert (ImageMagick).

[1] Z vlastní zkušenosti např. program JPEGmini a Kakadu u bezeztrátových např. OptiPNG

PNG Jedná se o formát PNG. Tento formát byl pro každý obrázek zvlášť optimalizován pomocí programu optipng. Optimalizační úroveň byla nastavena na 7 (jde tedy o nejvyšší doporučenou).

JPEG-XR Jedná se o bezeztrátový JPEG-XR konvertovaný pomocí programu convert (ImageMagick).

JPEG-LS Bezeztrátový JPEG-LS vytvořený knihovnou CharLS.

WTlib Jedná se o bezeztrátový formát prezentovaný touto prací. Díky různým nastavením umožňuje kompresi pomocí mnoha způsobů, a proto je nutné dále tato nastavení odlišovat.

WTlib A Jde o navrhované řešení pro práci s obrázky, které nejsou počítačem generované.

Barevná transformace	`RCT`
Dekompozice obrazu	`T[`∞`,0]` dokud je velikost alespoň 17 bodů
Vlnka	`5/3*`
Prediktor	Nebyl použit
Entropický kodér	EBCOT; LL, LH, HL mapovaní kontextů

WTlib MED Počítačem generované obrázky vykazují významné odlišnosti, díky kterým je vhodné s těmito druhy obrázků pracovat odlišným způsobem. Tato možnost proto využívá prediktor MED spolu s EBCOTEm.

Barevná transformace	`YCoCg-R`
Prediktor	`MED`
Vlnková transformace	Nebyla použita
Entropický kodér	EBCOT jako LL pásmo

3.6.2 Výsledky

Je složité předem odhadovat, jaký formát v jakém datasetu bude nejlepší. Podle autorů by měl JPEG-XR být zanedbatelně horší než JPEG 2000. Oba využívají techniky vhodné především pro běžné obrázky. Oproti tomu, PNG a JPEG-LS by měli být nedostižní pro obrázky počítačem generované.

Srovnávat předem PNG/JPEG-LS s WTlib MED není vůbec jednoduché, protože z těchto formátů WTlib MED nevychází. WTlib A by snad měl být lepší než JPEG 2000, protože byli zvažovány do jisté míry shodné metody komprese.

`Nastavení`	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
`PNG`	10,113	6,4436	11,222	10,463	1,2566
`JPEG-XR`	9,1056	6,7622	10,611	9,7719	4,1386
`JPEG-LS`	9,9680	12,532	10,900	10,663	1,7280
`JPEG 2000`	7,6590	**6,4034**	9,4033	8,7578	1,8736
`WTlib A`	**7,4846**	**6,3841**	**9,2602**	**8,6759**	1,7151
`WTlib MED`	7,8855	**6,3503**	9,7091	8,8910	**1,1190**

Tabulka 3.24: Různé formáty porovnané s navrhovanými řešeními

Hodnocení kompresního poměru

Běžné obrázky Z aktuálních řešení je jednoznačná převaha JPEGu 2000. Zajímavý je především proklamovaný minimální rozdíl ve výkonu JPEG XR v porovnání JPEGem 2000 se skutečnými výsledky. Pravděpodobně tím autoři mysleli především ztrátovou variantu tohoto formátu, protože u bezeztrátové byl zaznamenán výrazný rozdíl (cca 1111 kB na obrázek v setu kodak).

JPEG LS a PNG se očekávaně umístily u běžných obrázků na posledních místech (oba formáty používají prediktory).

Porovnání JPEGu 2000 a WTlib A vychází stabilně ve prospěch WTlib A, i když nelze mluvit o velikém rozdílu (134 kB na obrázek pro set kodak). Podstatné je ovšem to, že WTlib A je teoreticky rychlejší (např. dekompozice ušetří 30 % prostředků) a méně komplexní (např. absence mapování kontextů HH pásem) při zachování většiny možností, jaké nabízí JPEG 2000.

WTlib MED, která byla uzpůsobena především pro počítačem generované obrázky, je lepší než JPEG-LS, JPEG-XR a PNG.

Počítačem generované obrázky Pro tento druh obrázků by měl být vhodný formát PNG nebo JPEG LS. PNG zřejmě kvůli zastoupení fotografií s několika málo barvami v porovnání zvítězilo. Otázkou je, jestli není vhodnější na tyto druhy použít vektorovou grafiku.

WTlib A se drží na úrovni JPEGu LS. WTlib MED, který by měl být pro tento druh vhodnější díky použití prediktoru, se zdá být lepší než PNG.

Hodnocení rychlosti

Rychlost zpracování může být pro některé účely důležitá, avšak nelze jednoduše říct, že některý formát je rychlejší než jiný. Roli zde hraje ochota vývojáře daného programu optimalizovat, paralelizovat a nést odpovědnost spojenou s chybami, které tyto techniky vývoje často doprovází. Pokud má navíc cílová platforma nějaká technologické omezení (např. ASIC, FPGA, nějaký kvantový nebo jednojádrový počítač, mikročip, ...), mohou se váhy obrátit na zcela opačnou stranu. Dále moderní formáty mohou profitovat z instrukcí SIMD (které jsou na aktuální testovací platformě omezené). Z tohoto důvodu by bylo nešťastné v tomto směru kategorizovat formát bez důkladného seznámení se s jeho implementací a možnostmi portované platformy.

Následující porovnání formátů tedy nemůže být jednoznačné. WTlib na rozdíl od ostatních formátů nemusí splňovat tak přísné požadavky[2], a tím může být zápis rychlejší. Na druhou stranu je WTlib konstruován do co největší flexibility, a tak je navržen jako neoptimální (např. mnoho průchodů WT by bylo možné provádět in situ).

Následující test konvertoval běžný obrázek 3072×2048 na procesoru Intel Core i7 950, SSD, OS Fedora 25.

[2] Například rychlé vyhledávání informace v záznamu

Formát	Počet aktivních vláken	Rychlost (s)	Vstupní formát
PNG	1(?)	4.5	PPM
JPEG XR	1(?)	4.5	PNG
JPEG LS	1(?)	4.1	PPM
JPEG 2000	1(?)	3.9	PNG
WTlib A	1	3.6	PNG
WTlib MED	1	2.9	PNG
WTlib A	16 (EBCOT)	1.8	PNG
WTlib MED	16 (EBCOT)	1.6	PNG

Tabulka 3.25: Rychlost zpracování jednoho běžného snímku

Zajímavé bylo, že komprese formátu JPEG 2000 byla rychlejší než JPEG-LS. WTlib je mírně rychlejší než všechny aktuální formáty. WTlib s paralelizovaným EBCOTem odhaluje jeho výrazný podíl práce uvažujeme-li, že konstantou je celé dekódování formátu PNG, vlnková transformace a režije disku.

Kapitola 4

Závěr

Vlnková transformace je velmi výkonný nástroj pro odstraňování redundance z obrazových dat. Vyniká svojí jednoduchostí, rychlostí a při správně zvoleném entropickém kódování i kvalitou výstupu. Umožňuje splnit i moderní požadavky obrazových formátů, a to bez postihu v komprimačním poměru. Díky předchozím vlastnostem zatím překonává další zkoumané techniky zavedené konkurenčními bezeztrátovými formáty – a to jak prediktory (JPEG-LS, PNG), tak různými aproximace diskrétní kosinové transformace (JPEG XR). Její implementaci obsahuje populární formát JPEG 2000.

Vlnkovou transformaci lze implementovat nevyčíslitelně mnoha způsoby, proto je nemožné jednoduše zjistit, který způsob je nejvhodnější. V této práci byly různým testům podrobeny obě nezávislé součásti vlnkové transformace – rozklad obrazu a vlnky. K oběma částem byly navrženy úpravy, které oproti JPEGu 2000 mírně zlepšují bezeztrátový kompresní poměr, rychlost a komplexnost celé komprese.

Počítačem generované obrázky se liší v přesnosti přechodů a absencí šumu na plochách. Pro ně bylo zjištěno, že ani jedna ze zkoumaných modifikací vlnkové transformace nedosahuje výkonu jako MED prediktor zkombinovaný s EBCOTem, čili algoritmem pro entropické kódování pásem z vlnkové transformace. Tato souhra dvou technik dokonce významně překonala formát PNG optimalizovaný pro každý obrázek zvlášť. Díky použití prediktoru zbývá vyřešit problém v aktuálním požadavku vyžadujícím rychlé zobrazování náhledů bez úplné dekomprese. Jeho modifikace by proto mohly být vhodné např. pro webové služby, kde tento požadavek není nutný a je kladen důraz na minimální velikosti souborů.

Kromě dvou hlavních kategorií obrázků (reálné a umělé) nebyla nalezena žádná další kategorie, u které by se výrazně vyplatilo použití jiných kompresních technik. Nelze však očekávat stejné chování i u typů ze speciálních prostředí jako jsou vesmír, endoskopie, podmořské, . . . , u kterých lze očekávat minimálně rozdíl ve výhodnosti barevné transformace.

Bylo porovnáno velké množství vlnek se společným výsledkem: Jednodušší bývají výhodnější. Vzhledem k této informaci je pravděpodobné, že hledání lepší vlnky nebude výrazným zlepšením. Podobný vztah platil i pro rozklad obrazu. I přes toto zjištění by bylo nešťastné tvrdit, že je v tomto směru práce dokončena. Z chování této transformace lze totiž usoudit, že hledání optimálního řešení leží pravděpodobně v souladu mezi rozkladem obrazu a entropickým kódováním. Konkrétně u kódování by mohlo být výhodné rozpoznávání více kontextů, kódování větších bloků nebo využívání odpovídajících informací z vyšších pásem. U počítačem generovaných obrázků by bylo vhodné zvážit jiná řešení zrychleného kódování více bitů v jednom kroku a s tím související změnu průchodu aktuální oblasti.

Pod licencí LGPL byla vytvořena knihovna a program v jazyce C++, který je součástí přiloženého CD. Kopie je uložená na adrese https://github.com/tumpji/WTlib.

Literatura

[1] About YUV Video. [Online][cit. 2015-11-20].
URL
https://msdn.microsoft.com/en-us/library/windows/desktop/bb530104.aspx

[2] Border Effects. [Online][cit. 2017-5-1].
URL https:
//www.mathworks.com/help/wavelet/ug/dealing-with-border-distortion.html

[3] Digital Signal Processing Library. [Online][cit. 2017-1-17].
URL http://en.dsplib.org/index.html

[4] The Fast Lifting Wavelet Transform. [Online][cit. 2017-2-3].
URL
http://www.polyvalens.com/blog/wavelets/fast-lifting-wavelet-transform/

[5] FFmpeg – zdrojové kódy. [Online][cit. 2017-1-20].
URL https://ffmpeg.org/download.html#get-sources

[6] jxrlib 1.1. [Online][cit. 2017-4-14].
URL https://jxrlib.codeplex.com/SourceControl/latest

[7] A Linear Algebra View of the Wavelet Transform. [Online][cit. 2017-2-03].
URL http://www.bearcave.com/misl/misl_tech/wavelets/matrix

[8] A really friendly guide to wavelets. [Online][cit. 2017-1-17].
URL http://www.polyvalens.com/blog/wavelets/theory/#7

[9] Adams, M. D.; Kossentini, F.: Reversible Integer-to-Integer Wavelet Transforms for Image Compression: Performance Evaluation and Analysis. *IEEE Transactions on Image Processing*, ročník 9, 2000: s. 1010–1024.

[10] Akansu, A. N.; Smith, M. J. T.: *Subband and wavelet transforms*. Kluwer academic publichers, 1996, ISBN 0-7923-9645-6.

[11] Avramović, A.: Lossless Compression of Medical Images Based on Gradient Edge Detection.
URL https://etfbl.net/~aleksej/compress/telfor2011.pdf

[12] Bodden, E.; Clasen, M.; Kneis, J.: *Arithmetic Coding revealed*. 2004.

[13] Daubechies, I.; Sweldnes, W.: Factoring wavelet transforms into lifting steps.
URL http://cm-bell-labs.github.io/who/wim/papers/factor/factor.pdf

[14] David Salomon, M. G.: *Handbook of data compression*. Springer, 2010, ISBN 978 1 84882 902 2.

[15] Dufaux, F.; Sullivan, G. J.; Ebrahimi, T.: The JPEG XR Image Coding Standard. *IEEE signal processing magazine*, ročník 26, 2009: s. 195–204.

[16] Jančovič, A.: *Vnímání barev*. Diplomová práce, Masarykova univerzita v Brně, Fakulta pedagogická, 2005, [Online][cit. 2016-12-12].
URL http://www.ped.muni.cz/wphy/publikace/jancovic1.html

[17] Liang, J.; Tran, T. D.: Fast Multiplierless Approximations of the DCT With the Lifting Scheme. *IEEE Transactions on signal processing*, ročník 49, 2001: s. 3032–3044.

[18] Mallat, S. G.: A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transactions on pattern analysis and machine intelligence*, ročník 11, 1989: s. 674–693.

[19] Malvar, H.; Sullivan, G.: YCoCg-R: A Color Space with RGB Reversibility and Low Dynamic Range. [Online][cit. 2017-4-14].
URL https://www.microsoft.com/en-us/research/wp-content/uploads/2016/06/Malvar_Sullivan_YCoCg-R_JVT-I014r3-2.pdf

[20] Oppenheim, A. V.; Schafer, R. W.: *Discrete-Time Signal Processing*. Prentice-Hall, Inc., 1999, ISBN 0-13-754920-2.

[21] Road, L.; Wuhan: Boundary Effect Reduction in Wavelet Transform for Time-frequency Analysis. [Online][cit. 2017-5-1].
URL http://www.wseas.org/multimedia/journals/signal/2012/53-661.pdf

[22] Roelofs, G.: *PNG The Definitive Guide*. O'REILLY.
URL http://www.libpng.org/pub/png/book/cover.html

[23] Starosolski, R.: New simple and efficient color space transformations for lossless image compression. *Journal of Visual Communication and Image Representation*, ročník 25, 2014: s. 1056–1063.
URL http://www.sciencedirect.com/science/article/pii/S1047320314000595

[24] Taubman, D. S.; Marcellin, M. W.: *JPEG 2000*. Kluwer academic publishers, 2002, ISBN 0-7923-7519-X.

[25] Theodoridis, S.; Koutroumbas, K.: *Pattern Recogninion*. 2009, ISBN 978-1-59749-272-0.

[26] Uytterhoeven, G.; Bultheel, A.: The Red-Black Wavelet Transform.
URL https://nalag.cs.kuleuven.be/papers/ade/redblack/preprint.pdf

[27] Uzel, P.: *Entropické kodéry*. Diplomová práce, Univerzita Karlova v Praze, Matematicko-fyzikální fakulta, 2009.

[28] Vrána, J.; Čermák, J.; Zezula, R.: Výpočet vlnkové transformace pomocí algoritmu lifting.
URL www.elektrorevue.cz/clanky/04034/index.html

[29] Weinberger, M. J.; Seroussi, G.: The LOCO-I Lossless Image Compression Algorithm: Principles and Standartization into JPEG-LS.
URL http://www.labs.hp.com/research/info_theory/loco/HPL-98-193R1.pdf

[30] White, R. L.; Greenfield, P.: A Scheme for Compressing Floating-Point Images. [Online][cit. 2017-1-14].
URL http://www.adass.org/adass/proceedings/adass98/whiterl/

[31] Wu, X.; Memon, N.: CALIC – A context based adaptive lossless image codec.
URL https://isis.poly.edu/memon/pdf/1996_calic.pdf

[32] Zheng, J.; Fang, J.; Han, C.: The selection of reversible integer-to-tinteger wavelet transforms for DEM multi-scale representation and progressive compression. *ISPRS*, ročník 38, 2008: s. 709–714.

Přílohy

Příloha A

Definice použitých barevných transformací a vlnek

A.1 Barevné transformace

V této části přílohy se nacházejí definice převodů barevných prostorů. Všechny převody mají jako vstup nebo výstup model RGB.

Vstupní a výstupní model RGB má hloubku barev velikosti m. Tato hodnota zároveň definuje obor hodnot barevných složek různých modelů.

$$R, G, B \in \langle 0, 2^m - 1 \rangle$$

V některých výpočtech se vyskytuje operace $mod(a)$, ta odpovídá klasickému zbytku po dělení s pevným dělitelem 2^m.

$$mod(a) = a \; mod \; 2^m$$

Operace $smod(a)$ je symetrické modulo. Jeho výhoda oproti klasickému zbytku po dělení spočívá v redukci vytváření ostrých hran u barevně souvisejících pixelů.

$$smod(a) = mod\left(a + 2^{m-1}\right) - 2^{m-1}$$

V následujícím výpisu je v prvním sloupci převod z RGB do jiného předem specifikovaného barevného prostoru, ve druhém sloupci je převod zpětný. Poslední informuje o rozsahu hodnot.

Všechny barevné transformace pocházejí z následujících zdrojů: [23, 29, 24].

RCT

$$\begin{array}{llll} Y = \left\lfloor \dfrac{R + 2G + B}{4} \right\rfloor & G = Y - \left\lfloor \dfrac{C_u + C_v}{4} \right\rfloor & & Y \in \langle 0, 2^m - 1 \rangle \\ C_u = B - G & R = C_v + G & & Cu, Cv \in \langle -2^m + 1, 2^m - 1 \rangle \\ C_v = R - G & B = C_u + G & & \end{array}$$

YCoCg-R

$$\begin{array}{llll} Y = t + \lfloor Cg/2 \rfloor & R = B + Co & & \\ C_o = R - B & G = Cg + t & & Y, t \in \langle 0, 2^m - 1 \rangle \\ C_g = G - t & B = t - \lfloor Co/2 \rfloor & & Co, Cg \in \langle -2^m + 1, 2^m - 1 \rangle \\ t = B + \lfloor Co/2 \rfloor & t = Y - \lfloor Cg/2 \rfloor & & \end{array}$$

LDgEb

$$\begin{array}{ll} L = R - \lfloor {}^{Dg}/_{2} \rfloor & R = L + \lfloor {}^{Dg}/_{2} \rfloor \\ Dg = R - G & G = R - Dg \\ Eb = B - L & B = Eb + L \end{array} \qquad \begin{array}{c} R, L \in \langle 0, 2^m - 1 \rangle \\ Dg, Eb \in \langle -2^m + 1, 2^m - 1 \rangle \end{array}$$

mLDgEb

$$\begin{array}{lll} mL = mod(R - \lfloor {}^{mDg}/_{2} \rfloor) & R = mod(mL + \lfloor {}^{mDg}/_{2} \rfloor) & mL \in \langle 0, 2^m - 1 \rangle \\ mDg = smod(R - G) & G = mod(R - mDg) & mDg \in \left\langle -2^{m-1}, 2^{m-1} - 1 \right\rangle \\ mEb = smod(B - mL) & B = mod(mEb + mL) & mEb \in \left\langle -2^{m-1}, 2^{m-1} - 1 \right\rangle \end{array}$$

RDgDb

$$\begin{array}{ll} R = R & R = R \\ Dg = R - G & G = R - Dg \\ Db = G - B & B = G - Db \end{array} \qquad \begin{array}{c} R, \in \langle 0, 2^m - 1 \rangle \\ Dg, Db \in \langle -2^m + 1, 2^m - 1 \rangle \end{array}$$

mRDgDb

$$\begin{array}{ll} R = R & R = R \\ mDg = smod(R - G) & G = mod(R - mDg) \\ mDb = smod(G - B) & B = mod(G - mDb) \end{array} \qquad \begin{array}{c} mDg \in \left\langle -2^{m-1}, 2^{m-1} - 1 \right\rangle \\ mDb \in \left\langle -2^{m-1}, 2^{m-1} - 1 \right\rangle \end{array}$$

LOCO-I

$$\begin{array}{ll} G = G & R = Dg + G \\ Dg = R - G & G = G \\ Cu = B - G & B = Cu + G \end{array} \qquad \begin{array}{c} G \in \langle 0, 2^m - 1 \rangle \\ Dg, Cu \in \langle -2^m + 1, 2^m - 1 \rangle \end{array}$$

A.2 Vlnky

Všechny mnou použité vlnky využívají liftingu, jednotlivé kroky lze nalézt na samostatných řádcích. Uvádím zde pouze dopřednou transformaci, protože zpětná je otázkou pouhého obrácení pořadí operací a převrácení znamének.

Jednotlivé vlnky, resp. kroky liftingu, jsem převzal z [9, 32]. Vlnky `5/3*`, `1/3*`, `1/7`, `1/7*`, `9/7-M*` jsem vytvořil úpravou podobně označených originálů sám.

`5/3`
$$\begin{cases} d[n] = d[n] - \left\lfloor \frac{1}{2}\left(s[n] + s[n+1]\right)\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1}{4}\left(d[n-1] + d[n]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

`5/3*`
$$\begin{cases} d[n] = d[n] - \left\lfloor \frac{1}{2}\left(s[n] + s[n+1]\right) + \frac{1}{2}\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1}{4}\left(d[n-1] + d[n]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

`9/7-F`
$$\begin{cases} d[n] = d[n] - \left\lfloor \frac{203}{128}\left(s[n] + s[n+1]\right) + \frac{1}{2}\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{-217}{4096}\left(d[n-1] + d[n]\right) + \frac{1}{2}\right\rfloor \\ d[n] = d[n] + \left\lfloor \frac{113}{128}\left(s[n] + s[n+1]\right) + \frac{1}{2}\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1817}{4096}\left(d[n-1] + d[n]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

`9/7-M`
$$\begin{cases} d[n] = d[n] + \left\lfloor \frac{1}{16}\left(\left(s[n-1] + s[n+2]\right) - 9\left(s[n] + s[n+1]\right)\right) + \frac{1}{2}\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1}{4}\left(d[n-1] + d[n]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

`9/7-M*`
$$\begin{cases} d[n] = d[n] + \left\lfloor \frac{1}{16}\left(\left(s[n-1] + s[n+2]\right) - 9\left(s[n] + s[n+1]\right)\right) - \frac{1}{2}\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1}{4}\left(d[n-1] + d[n]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

`5/11-C`
$$\begin{cases} d[n] = d[n] - \left\lfloor \frac{1}{2}\left(s[n] + s[n+1]\right)\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1}{4}\left(d[n-1] + d[n]\right) + \frac{1}{2}\right\rfloor \\ d[n] = d[n] + \left\lfloor \frac{1}{16}\left(s[n-1] - s[n] - s[n+1] + s[n+2]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

`5/11-A`
$$\begin{cases} d[n] = d[n] - \left\lfloor \frac{1}{2}\left(s[n] + s[n+1]\right)\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1}{4}\left(d[n-1] + d[n]\right) + \frac{1}{2}\right\rfloor \\ d[n] = d[n] + \left\lfloor \frac{1}{32}\left(s[n-1] - s[n] - s[n+1] + s[n+2]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

`13/7-T`
$$\begin{cases} d[n] = d[n] + \left\lfloor \frac{1}{16}\left(s[n-1] + s[n+2] - 9\left(s[n] + s[n+1]\right)\right) + \frac{1}{2}\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1}{32}\left(9\left(d[n] + d[n-1]\right) - d[n-2] - d[n+1]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

`13/7-C`
$$\begin{cases} d[n] = d[n] + \left\lfloor \frac{1}{16}\left(s[n-1] + s[n+2] - 9\left(s[n] + s[n+1]\right)\right) + \frac{1}{2}\right\rfloor \\ s[n] = s[n] + \left\lfloor \frac{1}{16}\left(5\left(d[n] + d[n-1]\right) - d[n-2] - d[n+1]\right) + \frac{1}{2}\right\rfloor \end{cases}$$

$$\texttt{2/6}\begin{cases} d[n] = d[n] - s[n] \\ s[n] = s[n] + \left\lfloor \frac{1}{2}\left(d[n]\right) \right\rfloor \\ d[n] = d[n] + \left\lfloor \frac{1}{4}\left(s[n-1] - s[n+1]\right) + \frac{1}{2} \right\rfloor \end{cases}$$

$$\texttt{2/10}\begin{cases} d[n] = d[n] - s[n] \\ s[n] = s[n] + \left\lfloor \frac{1}{2}\left(d[n]\right) \right\rfloor \\ d[n] = d[n] + \left\lfloor \frac{1}{64}\left(22(s[n-1] - s[n+1]) + 3(-s[n-2] + s[n+2])\right) + \frac{1}{2} \right\rfloor \end{cases}$$

$$\texttt{6/14}\begin{cases} d[n] = d[n] - s[n] \\ s[n] = s[n] + \left\lfloor \frac{1}{16}\left(d[n-1] + 8d[n] - d[n+1]\right) + \frac{1}{2} \right\rfloor \\ d[n] = d[n] + \left\lfloor \frac{1}{64}\left(-s[n-2] + s[n+2] + 6(s[n-1] - s[n+1])\right) + \frac{1}{2} \right\rfloor \end{cases}$$

$$\texttt{Haar}\begin{cases} d[n] = d[n] - s[n] \\ s[n] = s[n] + \left\lfloor \frac{1}{2}\left(d[n]\right) \right\rfloor \end{cases}$$

$$\texttt{1/3}\left\{ d[n] = d[n] - \left\lfloor \frac{1}{2}\left(s[n] + s[n+1]\right) \right\rfloor \right.$$

$$\texttt{1/3*}\left\{ d[n] = d[n] - \left\lfloor \frac{1}{2}\left(s[n] + s[n+1]\right) + \frac{1}{2} \right\rfloor \right.$$

$$\texttt{1/7}\left\{ d[n] = d[n] + \left\lfloor \frac{1}{16}\left(\left(s[n-1] + s[n+2]\right) - 9(s[n] + s[n+1])\right) + \frac{1}{2} \right\rfloor \right.$$

$$\texttt{1/7*}\left\{ d[n] = d[n] + \left\lfloor \frac{1}{16}\left(\left(s[n-1] + s[n+2]\right) - 9(s[n] + s[n+1])\right) - \frac{1}{2} \right\rfloor \right.$$

Příloha B

Kompletní výsledky

Vlnka	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
5/3	7,5752	**6,3871**	9,3387	8,7171	1,7286
5/3*	**7,4848**	**6,3867**	**9,2605**	**8,6761**	1,7153
5/3**	**7,4848**	**6,3867**	**9,2605**	**8,6761**	1,7153
9/7	8,3139	6,4920	9,9297	9,2105	2,5244
Haarova	8,5082	6,5198	10,155	9,5265	1,5896
9/7-M	7,8401	6,4437	9,5419	8,8459	2,1633
9/7-M*	7,6363	6,4053	9,3850	8,6877	1,8619
5/11-C	7,7749	6,4524	9,4923	8,8300	2,1144
5/11-A	7,6824	6,4380	9,4142	8,8047	2,0311
13/7-T	7,8809	6,4399	9,5617	8,8667	2,2433
13/7-C	7,8909	6,4460	9,5652	8,8820	2,2616
2/6	8,2083	6,4449	9,8353	9,0869	2,0252
2/10	8,1719	6,4577	9,8107	9,0412	2,1353
6/14	8,2166	6,4697	9,8356	9,0737	2,2673
1/3	7,5643	6,4614	9,4256	8,7191	**1,5573**
1/3*	**7,4929**	6,4607	9,3617	8,6905	**1,5505**
1/7	7,8826	6,5876	9,7105	8,9421	2,0017
1/7*	7,7615	6,5550	9,6122	8,8231	1,7974

Obrázek B.1: Vliv různých vlnek na výkon komprese (rozklad `T[X,0]`)

Vlnka	Dataset (bpp)				
	Kodak	USC-misc	Přírodní	Zástavba	Poč. gen.
5/3	8,0474	6,6318	9,7503	9,1647	2,4206
5/3*	7,8757	6,6139	9,5918	9,0452	2,2799
5/3**	7,9239	6,6326	9,6379	9,0956	2,3487
9/7	8,5374	6,6954	10,125	9,4336	3,2262
Haarova	8,8909	6,8153	10,528	9,8931	2,3935
9/7-M	8,1458	6,6741	9,8252	9,1895	2,8958
9/7-M*	7,8698	6,6216	9,6033	8,9722	2,4622
5/11-C	8,0905	6,6835	9,7887	9,1863	2,8255
5/11-A	8,1019	6,6711	9,7861	9,2131	2,7176
13/7-T	8,1863	6,6753	9,8485	9,2074	3,0613
13/7-C	8,1928	6,6804	9,8488	9,2143	3,0847
2/6	8,7325	6,7962	10,350	9,6678	2,9056
2/10	8,7981	6,8480	10,413	9,7152	3,1169
6/14	8,7984	6,8615	10,414	9,7297	3,2427
1/3	7,8806	6,6159	9,6526	9,0212	2,1718
1/3*	**7,7815**	**6,6028**	**9,5654**	**8,9614**	**2,0415**
1/7	7,9962	6,7069	9,7582	9,1060	2,5852
1/7*	7,8774	6,6739	9,6626	8,9941	2,3686

Obrázek B.2: Vliv různých vlnkek na výkon komprese (červeno-černá WT)

Příloha C

Manuál programu

Popis rozhraní programu

Většinu důležitých informací se lze dočíst v nápovědě, která se vypíše pomocí přepínače `--help` nebo `-h`. Proto zde pouze doplním informace o přepínačích vyžadujících důkladnější popis.

Barevná transformace

Knihovna WTlib nabízí několik barevných transformací nebo možnost transformaci nepoužívat. Výběr těchto možností umožňuje přepínač `--color` nebo `-c`. Po něm následuje číslo indikující danou barevnou transformaci. To je zjistitelné z rychlé nápovědy `-h`. Dané řešení bylo zvoleno proto, aby bylo možné rychle přidávat další rozšíření bez změny kódu na mnoha místech. V případě, že data nebudou kompatibilní (např. černobílý obrázek), vytiskne se varování a pokračuje se bez transformace. Implicitní barevná transformace je do prostoru RCT.

Vlnky

Výběr aplikované vlnky v dekompozici lze specifikovat pomocí `--wavelet` nebo `-w`. Tyto vlnky lze používat bez ohledu na dekompozici nebo jiné nastavení transformace. Implicitní vlnka je `5/3`.

Rozklad obrazu

Knihovna WTlib nabízí velký výběr nastavení dekompozice obrazu.

`--dec_max_level`
: Udává maximální počet úrovní vybrané dekompozice. Přitom jako úroveň je v tomto případě chápáno jedno horizontální, vertikální, červeno-černé, modro-žluté, ... rozložení na dvě pásma.

  0 Nepoužívat vlnkovou transformaci.

  1 U separabilních dekompozic se jedná o jeden horizontální průchod. V případě červeno-černé vlnkové transformace se pásma rozloží na červené a černé hodnoty, které budou beze změny uloženy jako 4 pásma.

  2 U separabilní provede rozložení minimálně na LL,LH,H. U červeno-černé WT provede minimálně červeno-černý lifting a na červených vzorcích dále modro-žlutý.

  Implicitní je konstanta `INT_MAX` ($2.15 \cdot 10^9$), tzn. že se bude rozkládat až do úrovně jednoho pixelu.

`--dec_min_size`
: Udává minimální velikost pásma, které je ještě připuštěno k dalšímu rozkladu (bez ohledu na `dec_max_level` nebo `dec_side_level`).
  Implicitní je velikost 2.

`--decomposition, -d`
: Udává výběr hlavních chování dekompozice. Na výběr je dekompozice typu separabilní (pyramidová) a červeno-černá.
  Implicitní je separabilní dekompozice.

`--dec_side_level`
: Udává počet rozkládání vysokofrekvenčních vedlejších pásem v příslušné dekompozici. Tedy kolikrát jsou tato pásma dále rozkládána.

  Toto chování je ilustrováno při separabilní dekompozici na obrázku 3.11, kde odpovídá označení `p` v `T[X,p]`. Je zaručené, že vedlejší pásma budou minimálně tak veliká jako poslední LL.

  Červeno-černá dekompozice je rozkládána obdobně a i u ní v tomto smyslu platí konstrukční omezení, které je zmíněno zmíněné v prvním bodě.
  Implicitní je číslo 1 implementující klasickou podobu pyramidové dekompozice nebo variantu červeno-černé dekompozice s rozložením černého pásma na černo-modré a černo-žluté.

Prediktory

`--predictor, -p`
: Umožňuje použít prediktor ještě před použitím vlnkové transformace.

`--postpredict`
: Aplikace různých prediktorů po vlnkové transformaci na různá pásma. Předává se řetězec znaků. Každý znak reprezentuje jeden prediktor, ten je mapován stejně jako přepínač `-p`. O pásmu rozhoduje pořadí, takže první znak bude vztažen k LL, druhý k LH, HL, HH. Podobně je to i s červeno-černou dekompozicí: modré, žluté, černo-modré, černo-žluté.

Entropické kódování

Je implementován EBCOT s pevnou velikostí kódovaných bloků na 64×64. Alternativou k němu je dvojitý aritmetický kodér. Nejprve kóduje znaménka, využívá 32 kontextů vypočítaných z označení pásma, levého, horního a diagonálního znaménka ($\Sigma_{sign} = -, +$). Následně kóduje celé hodnoty, jeho abeceda je $\Sigma_{coef} = 0, \ldots, 2^N - 1, E$, kde N lze vybrat z 8, 9, 10 bitů. V případě, že tento kodér narazí na číslo, které nelze reprezentovat pomocí daného počtu bitů, je nejprve vloženo N dolních bitů, následně je vložen znak E, který značí pokračování. Po tomto znaku následuje stejným způsobem ukládání zbytkového bitově posunutého čísla (o N).

`--coder, -e`
: Pomocí tohoto přepínače lze nastavit implicitní chování. To znamená, jestli je použit EBCOT, tak pásma označená jako LL, LH, HL, HH se budou kódovat příslušnými kontexty. V případě červeno-černé dekompozice není žádné implicitní chování.

  Implicitní je EBCOT.

`--code_alt`
: Modifikuje implicitní chování a umožňuje zakódovat např. pásmo LL s kontextem HH nebo úplně jiným kodérem. Vstup má podobnou logiku jako přepínač `--postpredict`.

Příloha D

Obsah CD

- Zdrojové kódy vytvořené knihovy
- Zdrojové kódy ukázkového programu
- Licence
- Stručný návod k použití a překlad programu a knihovny