Projekt do předmětu ZPO 2013L
Komprimace obrazu waveletovou transformací

Jan Dušek
xdusek17@stud.fit.vutbr.cz

1.5.2013

1 Úvod

V tomto projektu máme za úkol provést komprimaci obrazu waveletovou transformací. Waveletová transformace sama o sobě nic nekomprimuje, musíme proto po ní aplikovat komprimační algoritmus, který komprimuje výsledné koeficienty a umožní nám některé koeficienty vypustit, takže nám umožní vytvořit komprimaci ztrátovou, díky které podstatně zmenšíme velikost obrazových dat. V tomto projektu jsme implementovali algoritmus EZW, který se přesně na tento problém specializuje. Výslednou komprimaci porovnáme s běžně používaným formátem JPEG.

2 Analýza řešení

V této kapitole si popíšeme některé algoritmy, jež jsou klíčové pro komprimaci obrazu pomocí waveletové transformace. Hlavní výhodou tohoto řešení je možnost provést komprimaci ztrátovou, kdy je možno vypustit velké množství informace aniž by obraz trpěl vysokou ztrátou kvality.

2.1 Diskrétní waveletová transformace

[2] Diskrétní waveletová transformace (DWT) je následníkem diskrétní kosinové transformace (DCT), která se používá v populárním formátu JPEG. V jeho nástupci JPEG2000 se používá již diskrétní waveletová tranformace, která dává lepší výsledky, její hlavní výhodou oproti DCT je, že zachytává jak frekvenci tak polohu.

DWT není jedna jediná transformace nýbrž rodina transformací, kdy její konkrétní podoba je dána vlnkou (waveletou) pomocí, které se transformuje. Existuje mnoho různých vlnek nejzákladnější je tzv. Haarova vlnka, která může být reprezentována takto:

$$\psi(t) = \begin{cases} 1 & 0 \leq t \leq \frac{1}{2} \\ -1 & \frac{1}{2} \leq t < 1 \\ 0 & \text{jinak} \end{cases} \tag{1}$$

Tato vlnka se málokdy přímo používá a vycházejí z ní všechny ostatní.

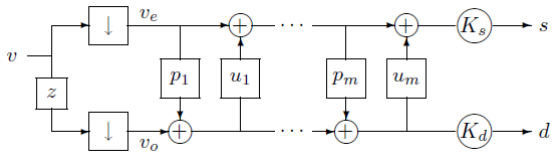
Transformaci lze implementovat několika způsoby, prvním je banka filtrů, kdy máme dva filtry h a g horní propusť respektive dolní propusť, přes které filtrujeme vstupní signál x a výstup pod-vzorkujeme, takto nám vzniknou druhy koeficientů, přibližné a detailní. Toto můžeme zapsat vzorci:

$$\begin{aligned} c[n] &= (x * h)[n] \downarrow 2 \\ d[n] &= (x * g)[n] \downarrow 2 \end{aligned}$$

kde c a d jsou přibližné respektive detailní koeficienty a $\downarrow$ je operátor pod-vzorkování. Pokud přivedeme přibližné koeficienty na vstup výše uvedené rovnice získáme tak dvojúrovňovou transformaci, takto lze postupovat až dosáhneme požadované úrovně.

Druhý způsob se nazývá Lifting scheme [6] je odvozen od banky filtrů a sestává z tzv. *predict* a *update* filtrů. Po aplikování filtrů je pak signál násoben tzv. *scale factorem*. Pro inverzní transformaci stačí obrátit pořadí filtrů a jejich znaménka. Na obrázku 1 je vidět Lifting scheme s *predict, update* filtry p_i respektive u_i a K_s, K_d *scale factory*.

Obrázek 1: Lifting scheme



Jelikož samotná transformace nijak nekomprimuje, z obrázku o rozměrech 512x512 získáme čtyři pole koeficientů o 256x256 velikosti tedy počet koeficientů je stejný jako počet pixelů obrázku, tak musíme použít nějaký algoritmus jež koeficienty zkomprimuje.

2.2 EZW kódování

[3] *Embedded Zerotree Wavelet* kódování je způsob kódování koeficientů vzniklých dvourozměrnou waveletovou transformací.

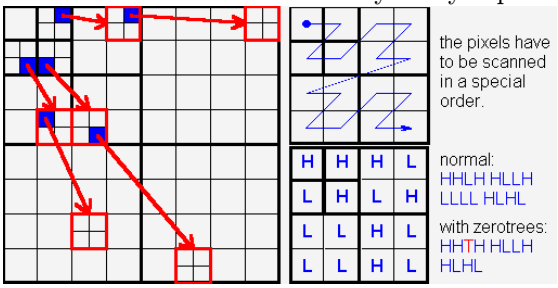
Základní vlastností tohoto algoritmu je, že patří do třídy progresivních kódování, tedy čím více dat z výstupu si uchováme tak je dekomprese přesnější. Pokud algoritmus necháme dokončit získáme bezeztrátovou kompresi, a pokud algoritmus ukončíme předčasně tak získáme ztrátovou kompresi, přičemž čímž ukončíme kompresi později tak získáme menší ztrátu informace. Progresivní kódování se někdy označuje jako *embedded*, což vysvětluje první písmeno E v názvu algoritmu.

EZW využívá dvou důležitých pozorování a to, že přírodní obrazy mají většinou více energie v nižších pásmech než vyšších, takže můžeme vyšší pásma pouze přidávají detail a můžeme je vynechat, a že vyšší koeficienty jsou důležitější než nižší. Koeficienty jsou tedy kódovány v sestupném pořadí, tak že jsou porovnávány se zvoleným prahem, pokud jsou vyšší tak jsou kódovány, pokud nižší tak jsou ponechány následujícímu průchodu. Po průchodu všemi koeficienty je snížen práh a všechny koeficienty jsou znovu kódovány a k obrazu je tedy přidáno více detailu. Tento proces pokračuje dokud nejsou všechny koeficienty zakódovány nebo je dosaženo např. maximálního bitového toku.

Trik spočívá ve využití závislostí mezi pásmy při kódování koeficientů menších jak práh. Waveletová transformace zachovává i pozici takže ji musíme nějak zakódovat. Koeficienty můžeme poskládat do stromu, kdy koeficient v nižším pásmu má čtyři potomky v pásmu vyšším. Tito potomci mají také čtyři potomky atd. čímž nám vzniká *quadtree*. Jelikož koeficienty klesají k vyšším pásmům definujeme tzv. *zerotree,* což je *quadtree* pro který platí, že všichni potomci jsou menší než rodič. Takový uzel lze pak zakódovat jediným symbolem, jelikož pokud je uzel nižší než práh pak i jeho potomci jsou nižší než práh. Ne všechny koeficienty budou splňovat tuto podmínku, ale v přirozeném obrázku budou takovéto koeficienty převažovat a my tak dosáhneme značné komprese.

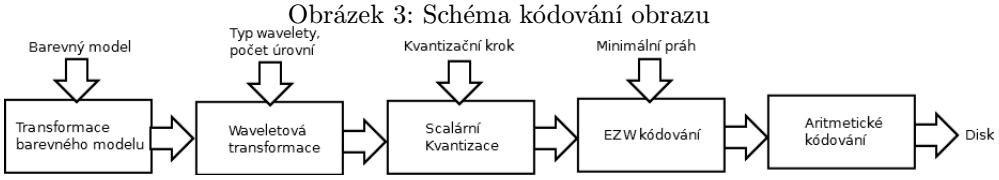
Na obrázku 2 lze vidět vztah mezi koeficienty různých pásem (vlevo), jak jimi procházet (vpravo nahoře) a použití *zerotree* (vpravo dole) symbolu T při kódování. Symbol H znamená, že koeficient je větší než práh a symbol L, že je menší.

Obrázek 2: Vztah mezi koeficienty různých pásem



Při implementaci tohoto posíláme na výstup symboly P (koeficient je větší než práh), N (menší než práh), T (koeficient je *zerotree* uzel) a Z (pokud je koeficient menší než práh ale není *zerotree* uzel). Zjištění zda je koeficient typu T je velice nákladné, jelikož musíme projít všechny uzly stromu a ověřit že jsou všechny splňují kritéria *zerotree*. Tyto symboly je možno kódovat nějakým statistickým kódováním pro další zlepšení komprese. V našem případě použijeme aritmetické kódování.

Obrázek 3: Schéma kódování obrazu



2.3 Aritmetické kódování

[7] Aritmetické kódování je druh statistického kódování, kdy celý text kódujeme jedním číslem z intervalu $[0, 1)$. Text je tvořen ze symbolů $a_1, a_2, \ldots, a_n$, které se vyskytují s pravděpodobnostmi $p_1, p_2, \ldots, p_n$. Principem je to, že se tento interval rozdělí na n disjunktních intervalů jejichž velikost je určena pravděpodobnostmi. Při zavedení kumulativních pravděpodobností

$$q_0 = 0,\ q_1 = p_1,\ q_2 = p_1 + p_2,\ q_n = p_1 + p_2 + \ldots + p_n = 1$$

patří symbolu a_i interval $[q_{i-1}, q_i)$. Postup aritmetického kódování je takový, že se nepočítá přímo výsledná hodnota, nýbrž se neustále vymezuje interval v němž výsledná hodnota leží, tento interval budeme v následujícím textu označovat symbolem I.

Nejdříve si zvolíme I=[0,1). Poté ze vstupu odebereme znak a_i, určíme jemu odpovídající interval $[q_{i-1}, q_i)$ a ze stávající hodnoty I=[l,h) vypočítáme novou dle vzorce:

$$I \quad = \quad [l + q_{i-1} * (h - l), l + q_i * (h - l))$$

tím jako nový interval vybereme tu část jež odpovídá intervalu znaku a_i. Předchozí krok opakujeme dokud nezakódujeme všechny znaky ze vstupu. Výsledkem pak libovolné číslo c z intervalu I.

Dekódování vychází ze stejného rozdělení na n disjunktních intervalů, v každém kroku pak nalezneme interval ve kterém se nachází zakódovaná hodnota c. Z něho určíme o který znak se jedná a vypočítáme novou hodnotu intervalu I stejným způsobem jako při kódování a pokračujeme v dekódování dalšího znaku.

Aritmetické kódování používá pro interpretaci intervalu I reálná čísla. S každým zakódovaným znakem se interval zmenší a čím menší tím je číslo potřebné k jeho zápisu delší. Princip je tedy takový, že znaky s větší pravděpodobností zmenšují interval méně než čísla s pravděpodobností menší. Jelikož práce s reálnými čísly je v počítači velmi složitá, tak reálné implementace používají celá čísla.

Z praktického hlediska je také důležité jak zjistíme pravděpodobnosti symbolů. Základní metodou je, že pravděpodobnosti známe již před započetím kódování a zůstávají po celou dobu komprese. Vzhledem k tomu, že jak kodér tak dekodér musí mít pravděpodobnosti stejné, je nutno tyto pravděpodobnosti přenášet spolu s hodnotou c. Druhou metodou je tzv. adaptivní kódování, kdy z počátku nastavíme pravděpodobnosti symbolů stejně, ale po každém zakódování znaku zvýšíme hodnotu pravděpodobnosti tohoto znaku. Dekodér s pravděpodobnostmi pracuje analogicky. a tedy není nutno si pravděpodobnosti nikam ukládat. Rovněž tímto adaptivní kódování bere v potaz změnu pravděpodobností různých znaků v průběhu textu. Vzhledem k tomu, že obrázky nejsou statisticky statické a pravděpodobnosti symbolů se mění region od regionu, použijeme adaptivní kódování, které může tyto změny dobře podchytit.

3 Návrh

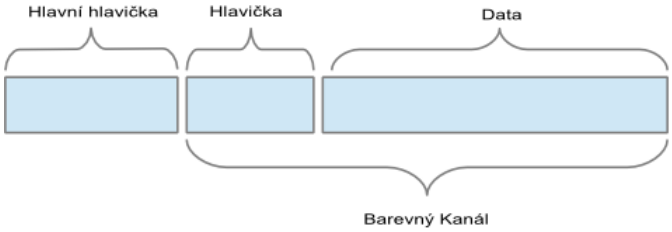
V této části si popíšeme návrh samotného systému pro komprimaci obrazu, pomocí waveletové transformace a všechny jeho kroky. Také si popíšeme formát dat jež byl vytvořen pro účel ukládání těchto komprimovaných dat na disk.

3.1 Návrh systému

Samotná komprimace sestává z několika dílčích kroků, ty můžeme vidět na obrázku 3.

Prvním krokem je transformace barevného modelu z BGR, ve kterém jsou data jež se mají zakódovat, do zvoleného barevného modelu ve kterém se bude dále pracovat. K dispozici je samozřejmě základní RGB, nebo formát YCbCr, který odděluje jasovou složku Y od barevných složek Cb a Cr. Jelikož oko je citlivější na jasovou složku než barevnou, je možno složky Cb a Cr podvzorkovat. Jistou nepříjemností je, že převod z BGR do YCbCr není reverzibilní pro převod se používají reálná čísla a zaokrouhlování, takže nelze s tímto barevným

Obrázek 4: Schéma formátu dat



modelem dosáhnou bezeztrátové komprese. Pro převod z BGR do YCbCr se používá následující vzorec [1]:

$$\begin{pmatrix} Y \\ C_b \\ C_r \end{pmatrix} = \begin{pmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Jednotlivé složky barevného modelu jsou následně zpracovávány samostatně.

Druhým krokem je samotná waveletová transformace, kdy si můžeme určit počet úrovní a typ wavelety, kterou použijeme. My použijeme wavelety z JPEG2000 a to Cdf 9/7 a Daub 5/3. Pro jejich výpočet využijeme Lifting scheme, kdy koeficienty pro Cdf 9/7 nalezneme v [6]. Daub 5/3 můžeme dle [8] vypočítat jen s použitím celých čísel pomocí vzorců:

$$\begin{aligned} y(2n+1) &= x(2n+1) - \left\lfloor \frac{x(2n) + x(2n+2)}{2} \right\rfloor \\ y(2n) &= x(2n) + \left\lfloor \frac{x(2n-1) + x(2n+1) + 2}{4} \right\rfloor \end{aligned}$$

Z toho vyplývá, že s Daub 5/3 lze dosáhnout bezeztrátové komprese, jelikož při použití celých čísel a předchozích vzorců lze původní signál zrekonstruovat. U Cdf 9/7 je nutno použít čísla s pevnou či plovoucí řádovou čárkou a u těch se zaokrouhlovacím chybám nevyhneme.

Po waveletové transformaci následuje skalární kvantizace, kdy na koeficienty aplikujeme vzorec:

$$k = \operatorname{sgn}(x) * \left\lfloor \frac{|x|}{\Delta} + \frac{1}{2} \right\rfloor$$

kdy x je vstup a Δ je kvantizační krok jež si zvolíme. Rekonstrukce posléze probíhá snadno $y = k * \Delta$. Výstupem kvantizace je vždy celé číslo a pokud je rovněž celé číslo vstupem a Δ=1 tak nedochází k žádné ztrátě informace a kvantizace je reverzibilní.

Pro EZW kódování v našem případě použijeme prahy jako mocniny dvou, takže budeme kódovat jednotlivé bitové roviny. Můžeme si zvolit minimální práh jež způsobí, že požadovaný počet nejméně významných bitových rovin nebude zakódován a dojde tak ke ztrátové kompresi dat. Pokud si minimální práh zvolíme nula, tak budou zakódovány všechny bitové roviny a bude se jednat o bezeztrátovou kompresi. Symboly z našeho EZW pošleme na vstup adaptivního aritmetického kodéru, jehož výstupem je již finální bitový proud.

3.2 Formát dat

V rámci tohoto projektu jsme navrhli formát na ukládání komprimovaných obrazových dat. Tento formát byl nazván WLF (*WaveLet Format*). Na obrázku 4 lze vidět jeho schéma, kdy máme jednu hlavní hlavičku souboru a pak barevné kanály, počet záleží na použitém barevném modelu typicky tři, které mají také vlastní hlavičku.

Hlavní hlavička obsahuje nejdříve osm magických bytů *0x89 0x57 0x4c 0x46 0x0d 0x0a 0x1a 0x 0a*. A dále šířku, výšku obrazu, barevný model, počet úrovní waveletové transformace, použitou waveletu a velikost kvantizačního kroku. Barevné kanály obsahují vlastní hlavičku s informacemi potřebnými pro EZW dekódování a to práh a minimální práh. Data jsou zakódována pomocí schématu uvedeném na obrázku 3.

4 Implementace

V této kapitole si popíšeme velmi zjednodušeně implementaci. Projekt byl rozdělen na několik částí. Hlavní částí je knihovna ve které je soustředěna veškerá funkcionalita. Dalšími částmi jsou programy pro konverzi z/do našeho WLF formátu, velmi jednoduchý prohlížeč WLF obrázků a program pro výpočet *PSNR* [5], který slouží k porovnávání kvalit obrázků komprimovaných ztrátovými metodami.

Obrázek 5: Referenční obrázek lena

4.1 Použité technologie

Projekt byl implementován v jazyce C++ s použitím nové normy C++11. Pro překlad a správu závislostí jsme použili multiplatformní *CMake*. Jedinou nestandardní knihovnou, která byla použita je *OpenCV* [4]. Tato knihovna je velmi obsáhlá, ale my jsme ji použili na práci s obrazovými soubory, zobrazování obrázků a převody barevných modelů.

Dále jsme při vývoji použili framework *googletest*, pro zjednodušené psaní jednotkových testů. Tento framework není k běhu programů vyžadován a lze projekt přeložit i bez něho.

4.2 Ovládání programů

V projektu byli vytvořeny tři programy. Prvním programem je *wlfconv*. Slouží ke konverzi z/do WLF formátu do/z formátů se kterými umí *OpenCV* manipulovat. Jeho použití je následující:

```
wlfconv [-f FORMAT -w WLET -l DWTLEVELS -c RATE -q STEP] INPUT OUTPUT
wlfconv -d INPUT OUTPUT
        -f FORMAT     barevný model, jeden z [rgb, ycbcr444(základ), ycbcr422]
        -w WLET       waveleta, jeden z [9/7(základ), 5/3]
        -l DWTLEVELS  počet úrovní waveletové transformace. (4)
        -c RATE       počet bitových rovin, které se vypustí (min. práh pro EZW) (0)
        -q STEP       kvantizační krok (1)
        -d            způsobí, že program konvertuje z WLF.
        INPUT         vstupní soubor. formát WLF pokud -d
        OUTPUT        výstupní soubor. formát WLF pokud není -d
```

Dalším programem je *wlfshow*. Slouží ke zobrazení obrázku ve formátu WLF. K zobrazení se použije *OpenCV* funkce *imshow*. Jeho použití je následující:

```
wlfshow IMAGE
        IMAGE         obrázek ve formátu WLF, který se zobrazí
```

Posledním programem je *psnr*. Slouží k výpočtu *PSNR*. Podporuje formáty *OpenCV* a náš formát WLF. Jeho použití:

```
psnr ORGINAL COMPRESSED
        ORGINAL       referenční obrázek
        COMPRESSED    obrázek kterému se spočítá PSNR vzhledem k referenčnímu obrázku.
```

5 Výsledky

Testování probíhalo na referenčním obrázku 5 o velikosti 512x512 bodů.

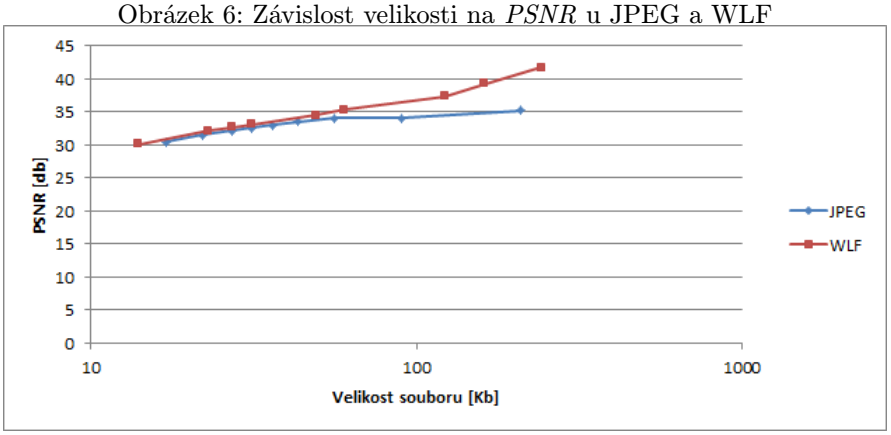
Nejprve srovnání bezeztrátové komprese našeho formátu WLF a formátu PNG. Bezeztrátová komprese je ve WLF dosažena pomocí wavelety Daub 5/3, RGB barevného modelu, kvantizačního kroku 1 a EZW minimálního prahu 0.

PNG	WLF
463 Kb	645 Kb

Tabulka 1: Naměřené hodnoty *PSNR* a velikostí

JPEG quality	Velikost [Kb]	*PSNR* [db]	*wlfconv* parametry	Velikost [Kb]	*PSNR* [db]
100	209	35,3768	-q 4	241	41,6708
90	90	34,1053	-c 2	162	39,2369
80	56	34,0102	-q 8	122	37,3193
70	43	33,4772	-c 3	60	35,3582
60	36	33,0135	-q 16	49	34,5012
50	31	32,5985	-q 24	31	33,1012
40	27	32,1571	-c 4	27	32,7131
30	22	31,5274	-q 32	23	32,1215
20	17	30,4397	-c 5	14	30,0972

Obrázek 6: Závislost velikosti na *PSNR* u JPEG a WLF



Jak je vidět, PNG ze srovnání vyšlo o něco lépe.

Dále porovnáme ztrátovou kompresi našeho formátu WLF a široce používaného JPEG, který používá Diskrétní kosinovou transformaci. V tabulce 1 vidíme naměřené hodnoty *PSNR* a velikostí formátu JPEG a WLF.

Tyto hodnoty vyvedeme do grafu 6 a vidíme, že náš formát lehce vede v oblasti nižších *PSNR*, a o dost vede v oblasti vyšších *PSNR*.

6 Závěr

V projektu se podařilo implementovat komprimaci pomocí waveletové transformace a algoritmu EZW. Byl vytvořen nový formát obrazových dat WLF určený k ukládání takto komprimovaných dat. K tomuto formátu byl vytvořen program jež dokáže konvertovat z/do tohoto formátu do/z běžných současně používaných formátů. Rovněž byl vytvořen základní prohlížeč obrázků v tomto novém formátu.

Nový formát dat prokázal lepší výsledky než běžně používaný JPEG, a to i přestože algoritmus EZW je pouze základní a nepříliš účinný. Existují novější a lepší algoritmy např. SPIHT. Rovněž algoritmus EZW trpí vysokou výpočetní náročností, mnohonásobně větší než jiné komprimační algoritmy ve formátech JPEG,PNG atd.

Reference

[1] [online]. [cit. 1.5.2013]. Dostupné na:
<`http://www.jpeg.org/.demo/FAQJpeg2k/functionalities.htm#What is the ICT?`>.

[2] *Discrete wavelet transform* [online]. [cit. 27.4.2013]. Dostupné na:
<`https://en.wikipedia.org/wiki/Discrete_wavelet_transform`>.

[3] *EZW Encoding* [online]. [cit. 1.5.2013]. Dostupné na:
<`http://www.polyvalens.com/blog/wavelets/ezw/`>.

[4] *OpenCV* [online]. [cit. 1.5.2013]. Dostupné na: <`http://opencv.org`>.

[5] *PSNR* [online]. [cit. 1.5.2013]. Dostupné na: <http://cs.wikipedia.org/wiki/PSNR>.

[6] GETREUER, P. *Filter Coefficients to Popular Wavelets.* 2006. Dostupné na: <http://www.mathworks.com/matlabcentral/fileexchange/5502-filter-coefficients-to-popular-wavelets>.

[7] VECERKA, A. *Komprese dat* [online]. 2008 [cit. 1.5.2013]. Dostupné na: <http://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf>.

[8] WALKER, J. S., NGUYEN, T. Q. a CHEN, Y.-J. A low-power, low-memory system for wavelet-based image compression. *Optical Engineering, Research Signposts.* 2003, roč. 5. S. 111–125.