Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Neztrátová komprese skupiny podobných obrazů

Plzeň, 2010 Karel Prouza

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne .....

Karel Prouza

Obsah

1. Úvod

Jak rychle se zvětšují velikosti pevných disků či záznamových médií, tak rychle se zvětšuje i objem dat, která chceme ukládat. Komprese dat je tedy jedním z nejdůležitějších problémů ve světě informatiky. U některých dat, například lékařských, je jejich zálohování doslova životně důležité a ušetřené místo po jejich kompresi může ušetřit cenné finanční zdroje. Zároveň je u takových dat nutné zachovat veškeré informace, které obsahují.

Existuje řada ztrátových či bezztrátových metod komprese dat. Některé metody jsou obecné, jiné jsou přizpůsobené pro kompresi konkrétního typu dat. Cílem této bakalářské práce je nejenom najít mezi bezztrátovými kompresními metodami vhodné metody pro kompresi grafických souborů, ale především využít vzájemné podobnosti komprimovaných obrázků pro dosažení vyššího kompresního poměru, než kdybychom tyto obrázky komprimovali samostatně.

V první části práce jsou uvedeny a popsány vybrané komprimační metody a grafické formáty, které tyto metody využívají. I když zadání práce hovoří o bezztrátové kompresi, jsou pro informaci uvedeny i některé ztrátové metody komprese. S grafickými formáty úzce souvisí barevné systémy, používané při ukládání grafických dat. Základní informace o barevných systémech jsou proto obsaženy v další části práce.

Při samotné realizaci komprimačního algoritmu musíme vyřešit několik problémů. Prvním z nich je jak odhalit podobnost u konkrétních obrázků a popsat, co to vlastně taková podobnost je. Dalším problémem je jak nalezenou podobnost co nejefektivněji využít a získat co největší kompresní poměr. Výsledná data samozřejmě musíme umět i rekonstruovat a uložit v nějakém z běžně používaných grafických formátů. Další část práce proto pojednává o použitých programových grafických knihovnách.

Ne všechna grafická data obsahují tolik podobnosti, aby u nich mohlo dojít k nějakému výraznému snížení velikosti po kompresi. Kromě výše zmíněných lékařských dat (konkrétně data z CT scanů) se nabízí i data, ve kterých na sebe obrázky nějak logicky navazují. Například jednotlivé snímky videa, či obrázky vycházející ze stejného zdroje vytvořené jeho úpravou. Porovnání dosažených kompresních poměrů u různých typů dat a různých typů kompresních metod je obsaženo v poslední části práce.

2. Komprese dat

Hlavním cílem komprese grafických nebo i jiných dat je maximální zmenšení velikosti dat a zároveň zachování všech informací. Toho se komprimační algoritmy snaží docílit především odstraněním redundance. Komprimovaná data se samozřejmě musí dát znovu obnovit na původní data. Kompresi dat můžeme rozdělit na dvě hlavní kategorie: bezztrátovou a ztrátovou kompresi.

2.1. Ztrátová komprese

Jak už z názvu vyplývá, hlavní myšlenkou této kategorie komprimačních metod je vypuštění některých „méně důležitých“ informací, které umožní větší efektivnost komprese. Kvalita komprese se dá u většiny algoritmů nastavit parametry. Vhodným nastavení kvality komprese můžeme dosáhnout lidskými smysly téměř neznatelného rozdílu a zároveň výrazného zmenšení objemu dat. Tento typ komprese se velmi často používá u zvukových nebo grafických souborů nebo u videa. Nejznámější ztrátové komprese jsou *MP3, OGG* nebo *WMA* u zvukových souborů a *MPEG* nebo *WMV* u video souborů. Nás ovšem nejvíce zajímají grafické soubory, kde jsou nejznámější ztrátové komprese *JPEG* nebo *JPEG2000.* Formát *JPEG2000* je popsán v podkapitole 2.2.5., protože umožňuje i bezztrátovou komprimaci, kterou ve svém programu využívám.

2.1.1. Formát JPEG

Nejznámější a nejpoužívanější grafický formát využívající ztrátovou kompresi je formát *JPEG* [SM1996]. Konkrétně využívá algoritmus DCT (*discrete cosine transformation*). Tento algoritmus přidává do obrázků redundanci, kterou mohou bezztrátové algoritmy efektněji zabalit. Komprimační poměr potom může být 20:1 až 30:1. K další ztrátě potom při převodu obrázku do formátu *JPEG* může dojít kvůli tomu, že formát *JPEG* používá barevný model YC_BC_R, který popisuje jiný barevný prostor než například model *RGB*.

2.2. Bezztrátová komprese

Bezztrátová komprese se používá v případě, kdy nechceme nebo si nemůžeme dovolit ztrátu žádné informace.

Jeden z nejznámějších algoritmů bezztrátové komprese je tzv. *Huffmannovo kódování*. Při tomto kódování se nejčastěji vyskytované hodnoty ukládají řetězcem s nejmenším počtem bitů a nejméně vyskytované hodnoty se naopak ukládají dlouhými bitovými řetězci.

Při bezztrátové komprimaci grafických souborů můžeme použít komprimační metodu RLE (run length encoding). Je to jednoduchá metoda, kdy se hodnoty vyskytující se několikrát za sebou uloží jako kombinace počtu výskytů a hodnoty.

Př: **3f 3f 3f 3f 3f** se uloží jako **05 3f**

3f 3f 3e 3f 3f se uloží jako **02 3f 01 3e 02 3f**

Je vidět, že pokud se hodnoty za sebou často střídají, efektivnost algoritmu výrazně ubývá. U toho algoritmu se dá i jednoduše vytvořit ztrátová verze. Pokud zaokrouhlíme u druhého příkladu prostřední hodnotu **3e** nahoru na **3f** výstup algoritmu by byl **05 3f**.

Metoda komprimace pomocí kvadrantového stromu rozděluje obrázek na čtyři části. Pokud je jedna z těchto částí vyplněna stejnou barvou, uloží hodnotu barvy. Pokud ne, rozdělí tuto část na další čtyři části. Takto rekurzivně pokračuje, dokud všechny části nejsou složeny ze stejných barev nebo dokud se nedostane až k samotnému jednomu pixelu.

2.2.1. Bzip2

Bzip2 je *open source* komprimační algoritmus vyvíjený Julianem Sewardem. Dosahuje lepšího kompresního poměru než například algoritmus *Deflate* používaný například u archivních formátů *ZIP*, ale je časově náročnější. Původně používal *Bzip* aritmetické kódování, které má lepší kompresní poměr, ale je zatíženo licencemi. Nyní proto *Bzip2* používá *Huffmanovo kódování.* Algoritmus *Bzip2* neumí pracovat s více

soubory, proto se při práci s více soubory tyto soubory nejprve spojí do jednoho například pomocí programu *tar* a teprve potom se provádí komprimace pomocí *Bzip2.*

2.2.2. LPAQ

LPAQ [M2010] je výrazně urychlená (až 35x) verze balícího algoritmu *PAQ* (který je velmi pomalý, ale ve většině testů má nejlepší kompresní poměr) ovšem za cenu nižšího kompresního poměru. Pro kompresi využívá *asymetrické binární kódování.* Stejně jako *Bzip2* umí pracovat pouze s jedním souborem. První verzi vydal Matt Mahoney v roce 2007, další verze vyvíjí Alexander Ratuschnyak, nejnovější je *LPAQ9m* z února 2009.

2.2.3. LZO

LZO [O2010] (Lempel-Ziv-Oberhumer) je knihovna pro bezztrátovou kompresi dat napsaná v ANSI C vyvíjená Markusem Oberhumerem. Rychlost komprese je srovnatelná s algoritmem *Deflate*, knihovna je ovšem zaměřená hlavně na vysokou rychlost dekomprese. Algoritmus je paměťově velmi málo náročný, pro dekompresi nepotřebuje žádnou paměť a pro kompresi pouze 64kB paměti. Umožňuje zvolit vyšší kompresi, která trvá déle a vyžaduje o 8kB paměti více, ale na rychlost pozdější dekomprese nemá vůbec žádný vliv.

2.2.4. Formát PNG

PNG (Portable Network Graphics) [ZBSF2004] je relativně mladý grafický formát, který byl vyvinut hlavně pro přenos bezztrátové grafiky po síti. Jeho vývoj začal po roce 1994, kdy firma CompuServe zpoplatnila licenci formátu *GIF*. Vznikl tedy jako svobodná vylepšená náhrada zastaralého formátu *GIF*. S tímto formátem má několik společných vlastností.

Oba tyto formáty využívají pro kompresi metodu *LZW* (pojmenována podle svých

tvůrců Lempela-Ziva a Welche). Dále používají prokládání obrazu, které je velmi praktické při použití těchto formátů na síti, protože už při přenesení části dat, může uživatel rozpoznat základní rysy obrázku a případně jeho stahování zastavit.

Novinkou formátu *PNG* je možnost předzpracování každého pixelu. Typů předzpracování je celkem 5 a všechny jsou neztrátové:

0) – je přímo uložena hodnota pixelu

1) - je uložen rozdíl s pixelem vlevo

2) - je uložen rozdíl z pixelem nahoře

3) - je uložen průměr pixelu a jeho sousedů vlevo a nahoře

4) - je uložena hodnota získaná z pixelu a jeho 3 sousedů

Pro každý řádek obrazu lze použít jinou z těchto metod a docílit tak optimální komprese.

Ve formátu *PNG* můžeme, stejně jako u formátu *GIF*, ukládat obrázky s barevnou paletou, která může mít až 256 barev. Ovšem výrazné vylepšení oproti staršímu *GIFu* je možnost ukládání obrázků bez palety v barevném režimu *truecolor*, kde je každý pixel reprezentován až 32 bity (24 bitů pro barvy a 8 bitů pro alfu).

Velkou výhodou formátu *PNG* je, že není zatížen žádnými licencemi, a tak se dá najít hodně knihoven pro jeho používání. Navíc je to jeden z formátů podporovaných operačním systémem Windows, který umožňuje jeho náhledy.

2.2.5. Formát JPEG2000

Formát *JPEG2000* [ZBSF2004] je nástupce formátu *JPEG* vytvořený *Joint Photographic Experts Group* v roce 2000 umožňující vyšší kvalitu obrazu při stejné velikosti souboru jako *JPEG*, ale hlavně umožňující bezztrátovou kompresi obrazu. Na rozdíl od původního formátu *JPEG*, který obraz rozděluje do čtverců o pevné velikosti 8x8 pixelů, formát *JPEG2000* pracuje s obrazem jako s celkem. Díky tomu může dosáhnout většího kompresního poměru než formát *JPEG* a zároveň tím eliminuje

výskyt rušivých makropixelů, které se u původního formátu kvůli rozdělení na části objevovaly.

Princip formátu *JPEG2000* je, že při předzpracování dat provádí transformaci pomocí funkcí nazývaných *vlnky* (*wavelet*). Opakované použití vlnkových transformací umožňuje uchovat obraz ve více rozlišeních a optimalizovat data pro postupný přenos po síti.

Při bezztrátové kompresi dosahuje lepších výsledků než formát *PNG* hlavně u fotografií, u obrázků s nižším počtem barev je výhodnější použít formát *PNG*.

Bohužel tento formát zatím není podporován webovými prohlížeči ani operačním systémem Windows, zřejmě kvůli tomu, že je zatížen licencemi.

3. Barevné systémy

Následující odstavce stručně uvedou problematiku různých barevných systémů používaných při ukládání grafických souborů. Informace jsou čerpány z knih [SM1996] a [ZPSF2004] není-li uvedeno jinak.

Základní informace, které se ukládají do grafických souborů jsou informace o barvě. U rastrových obrázků je to konkrétně informace o barvě pixelu. Co to vlastně je barva? Abychom mohli odpovědět na tuto otázku musíme nejprve říci něco o světle. Světlo je z fyzikálního hlediska elektromagnetické vlnění v oblasti 10^8 MHz a každá barva odpovídá nějaké konkrétní frekvenci. Lidské oko rozeznává barvy podle několika vlastností světelného záření:

1) **Barva** - je základním atributem světla a je určena jeho vlnovou délkou (lidské oko vnímá vlnové délky od 550nm do 720nm)
2) **Jas** - je přímo úměrný intenzitě světla
3) **Sytost** - je tím vyšší, čím je užší frekvenční spektrum světla
4) **Světlost** - určuje velikost achromatické složky ve světle s určitou dominantní frekvencí

Je technicky v podstatě nemožné vytvořit například monitor, který by uměl v každém bodě vysílat světlo jakékoliv vlnové délky a tím zobrazoval barvy, proto vznikly barevné systémy, které umožňují kombinací několika barevných složek vytvářet velké množství barev.

Barevné modely vlastně určují jakým způsobem a z čeho vznikne výsledná barva. Definujeme je tedy množinou základních barev, způsobem míchání a způsobem měnění barevných charakteristik. Existují dva základní typy míchání barev:

1) **Aditivní míchání** – každým přidáním další složky vznikne světlejší barva, přidáním všech složek vznikne bílá barva (např. model RGB)
2) **Subtraktivní míchání** - každým přidáním další složky vznikne tmavší barva, přidáním všech složek vznikne černá barva (např. model CMY)

Základní barevné modely i některé složitější si nyní ve stručnosti popíšeme.

3.1. RGB

Zřejmě nejrozšířenější počítačový barevný systém [MR1995]. V tomto systému se nové barvy vytváří *aditivním mícháním*, přidáváním jednotlivých složek k černé barvě vzniká světlejší barva. Použité složky v tomto systému jsou:

1) **R –** červená s vlnovou délkou 630nm
2) **G** – zelená s vlnovou délkou 530nm
3) **B** – modrá s vlnovou délkou 450nm

Tyto barvy byly vybrány, protože lidské oko má právě pro jejich vlnové délky nejlepší citlivost. Intenzita jednotlivých složek se pohybuje na intervalu <0,1>, který se nejčastěji převádí na interval celých čísel 0..255 při použití 8bitů na každou barevnou složku.

Jelikož systém obsahuje 3 barevné složky a každá z nich má 2 mezní hodnoty (0 a 1) získáváme v tomto modelu celkem 8 základních barev. Tyto barvy a jejich hodnoty jsou uvedeny v tabulce 3.2.

Barevný systém RGB je technicky orientovaný, dobře se používá pokud chceme výslednou barvu vytvořit světelnými zdroji (např. monitor počítače).

3.2. CMY

Zástupcem barevných systémů se *subtraktivním mícháním* barev je systém CMY. Nové barvy vznikají přidáváním barevných složek k bílé barvě, čímž vzniká stále tmavší barva. Použité složky v tomto systému jsou:

1) **C** – tyrkysová (cyan)
2) **M –** fialová (magenta)
3) **Y –** žlutá (yellow)

Toto míchání barev má blíž k lidským zkušenostem s mícháním barev, proto je tento způsob přirozenější než *aditivní míchání* [SM1996]. Barevný systém CMY se používá nejvíc v polygrafii při tisknutí barevných obrázků či fotek, kde se ještě přidává čtvrtá složka (blacK) a výsledný barevný systém se pak označuje jako CMYK.

Vzhledem k tomu, že v tomto barevném systému se barva vytváří kombinací 3 barevných složek se dvěma mezními hodnotami (stejně jako u RGB), má i tento barevný systém celkem 8 základních barev. V tabulce 3.2. jsou tyto barvy uvedeny i s jejich hodnotami v systémech RGB a CMY.

Barva	Cyan	Magenta	Yellow	Red	Green	Blue
bílá	0	0	0	1	1	1
žlutá	0	0	1	1	1	0
fialová	0	1	0	1	0	1
červená	0	1	1	1	0	0
tyrkysová	1	0	0	0	1	1
zelená	1	0	1	0	1	0
modrá	1	1	0	0	0	1
černá	1	1	1	0	0	0

Tabulka 3.2. - základní barvy barevných systémů RGB a CMY, hodnoty jsou na intervalu <0,1>

3.3. YCbCr

Předchozí barevné systémy byly technicky orientované, aby se pomocí nich daly jednoduše tisknout barvy na tiskárnách nebo zobrazovat na monitorech. Existují však i jiné barevné modely, které respektují způsob vnímání barev lidským okem. V lidském oku jsou totiž 2 typy receptorů:

1) **Tyčinky –** pokrývají celou sítnici a umožňují vnímat všeobecné obrazové informace jako je jas nebo obrysy. Jejich počet je přibližně 75-150 milionů a jsou přibližně 10x citlivější než čípky.

2) **Čípky –** jsou umístěny ve středu sítnice a jsou citlivé na barvy. Fungují ovšem pouze při achromatickém osvětlení (např. denní světlo). Dají se rozdělit do 3 skupin, podle toho na jaké vlnové délky jsou nejcitlivější. Jejich celkový počet je nižší než u tyčinek a je okolo 6-8 milionů, z toho zhruba 64% jsou čípky červené, 32% zelené a 2% modré.

Na základě těchto znalostí je definován barevný systém YC_BC_R, kde je vnímaný jas oddělený od dvou barevných signálů. **Y** je jasová složka a nabývá hodnoty z intervalu <0,1>. $\mathbf{C_B}$ je modrá složka a $\mathbf{C_R}$ je červená složka a nabývají hodnoty na intervalu <-0.5, 0.5>.

Převod ze systému *RGB* na systém YC_BC_R:

$$\mathbf{Y} = (0{,}299 * R) + (0{,}587 * G) + (0{,}114 * B)$$
$$\mathbf{C_B} = (-0{,}1687 * R) - (0{,}3313 * G) + (0{,}5 * B)$$
$$\mathbf{C_R} = (0{,}5 * R) - (0{,}4187 * G) - (0{,}0813 * B)$$

Převod ze systému YC_BC_R na systém *RGB*:

$$\mathbf{R} = Y + 1{,}402 * C_R$$
$$\mathbf{G} = Y - 0{,}344 * C_B - 0{,}714 * C_R$$
$$\mathbf{B} = Y + 1{,}772 * C_B$$

Při převodech jsou hodnoty *R*, *G* a *B* na intervalu <0,1>.

Tento barevný systém se používá například pro přenos televizního signálu, kde se dala v éře černobílých televizí využít vlastnost tohoto systému, že pokud chceme získat pouze s odstíny šedé, pracujeme pouze se složkou Y a ostatní složky ignorujeme.

V počítačích se s tímto systémem můžeme setkat při zápisu obrázků ve formátu JPEG. Nevýhodou tohoto systému ovšem je, že při jeho převodu na barevný systém RGB či zpět dochází ke ztrátě informace.

3.4. YCoCg

Tento barevný systém odstraňuje hlavní nevýhodu systému YC_BC_R a tou je, že při

konverzi na běžně používaný barevný systém *RGB* dochází ke ztrátě informace. Narozdíl od YC_BC_R systém YC_OC_G není založen na lidském vnímání barev, ale je přizpůsoben tak, aby umožnil bezztrátovou konverzi na systém *RGB*. Aby ovšem konverze byla vskutku bezztrátová, musíme na barevné složky použít o jeden bit víc.

Y je stejně jako u předchozího barevného systému jasová složka, $\mathbf{C_O}$ je tentokrát oranžová složka a $\mathbf{C_G}$ je zelená složka.

Převod ze systému *RGB* na systém YC_OC_G (s využitím pomocné proměnné *t*):

$\mathbf{C_O} = R - B$
$t = B + (C_O >> 1)$
$\mathbf{C_G} = G - t$
$\mathbf{Y} = t + (C_G >> 1)$

Převod ze systému YC_OC_G na systém *RGB* (s využitím pomocné proměnné *t*):

$t = Y - (C_G >> 1)$
$\mathbf{G} = C_G + t$
$\mathbf{B} = t - (C_O >> 1)$
$\mathbf{R} = C_O + B$

Při převodech jsou hodnoty *R*, *G* a *B* na intervalu <0,255>.

Tento barevný systém používají některé moderní video kodeky např. Dirac nebo H.264.

3.5. CIELUV

Aby bylo možno barvy nějakým způsobem exaktně vyjádřit byl v roce 1931 vytvořen mezinárodní standard základních barev, jehož součástí byl i tzv. chromatický diagram *CIEXYZ*, známý také jako *CIE 1931* (*CIE – Commision Internationale de l'Eclairage*, mezinárodní komise pro osvětlení). Viz obrázek 3.5.

V roce 1976 byl definovaný barevný prostor *CIELUV*, jinak známý i pod

označením *CIE 1976* (L^*, u^*, v^*). Viz obrázek 3.5. Tento nový barevný prostor rozkládá barvy rovnoměrněji než *CIE 1931*, což znamená, že vzdálenosti mezi dvojicí barev v tomto prostoru lépe odpovídá lidskému vnímání rozdílům intenzity mezi barevnými odstíny.

Převod ze systému *RGB* na systém *CIEXYZ*:

$$\mathbf{X} = 0{,}4125 * R + 0{,}3576 * G + 0{,}1804 * B$$
$$\mathbf{Y} = 0{,}2127 * R + 0{,}7151 * G + 0{,}0722 * B$$
$$\mathbf{Z} = 0{,}0193 * R + 0{,}1192 * G + 0{,}9502 * B$$

Zpětný převod ze systému *CIEXYZ* na systém *RGB*:

$$\mathbf{R} = 3{,}2405 * X - 1{,}5372 * Y - 0{,}4985 * Z$$
$$\mathbf{G} = -0{,}9693 * X + 1{,}8760 * Y + 0{,}0416 * Z$$
$$\mathbf{B} = 0{,}0556 * X - 0{,}2040 * Y + 1{,}0573 * Z$$

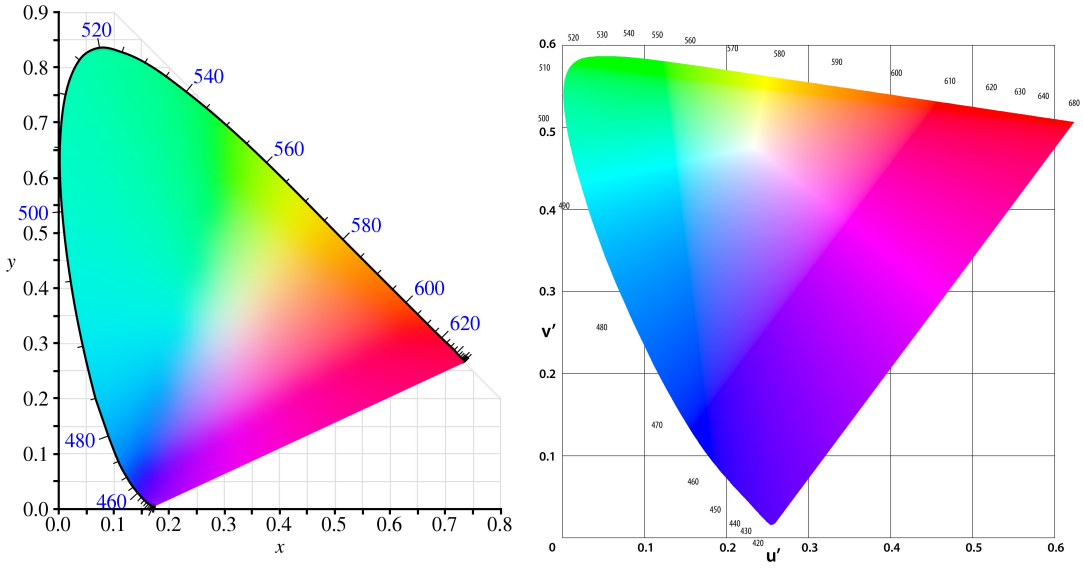
Při převodech jsou hodnoty *R*, *G* a *B* na intervalu <0,1>. Pokud při zpětném převodu vyjde některá z hodnot *R*, *G* nebo *B* mimo interval, jedná se o barvu, která nelze v barevném systému *RGB* zobrazit.

Převod ze systému *CIEXYZ* na systém *CIELUV*:

$$\mathbf{u'} = 4X / (X + 15Y + 3Z)$$
$$\mathbf{v'} = 9Y / (X + 15Y + 3Z)$$
$$\mathbf{L^*} = (116 * Y) - 16$$
$$\mathbf{u^*} = 13L^* * (u' - 0{,}2009)$$
$$\mathbf{v^*} = 13L^* * (v' - 0{,}4610)$$

Zpětný převod ze systému *CIELUV* na systém *CIEXYZ*:

$$\mathbf{u'} = u^* / (13L^*) + 0{,}2009$$
$$\mathbf{v'} = v^* / (13L^*) + 0{,}4610$$
$$\mathbf{Y} = ((L^* + 16) / 116)^3$$
$$\mathbf{X} = -9Yu' / ((u' - 4)\, v' - u'v')$$
$$\mathbf{Z} = (9Y - 15v'Y - v'X) / 3v'$$



Obrázek 3.5. - Porovnání barevných prostorů CIEXYZ (nalevo) a CIELUV (napravo)

zdroj: wikipedia.org

4. Způsoby porovnávání obrázků

Chceme-li porovnávat obrázky musíme nejprve definovat podle jakých kritérií budeme jejich podobnost hodnotit. Někdy potřebujeme obrázky setřídit podle toho, co zobrazují. Jindy je vhodnější třídit je podle toho jaké barvy obsahují nebo podle jejich kontrastu. V následujících odstavcích si představíme některé metriky, podle kterých se dají obrázky porovnávat.

4.1. PSNR

PSNR je zkratka z anglického *Peak Signal to Noise Ratio* a vyjadřuje poměr mezi maximálním možným signálem a šumem. Při porovnávání dvou obrázků touto metodou se spočítá střední kvadratická odchylka, což je součet kvadrátů rozdílu každých dvou sobě odpovídajících pixelů obrázků dělený celkovým počtem pixelů:

$$MSE = \frac{1}{xy} \times \sum_{i=0}^{x-1} \sum_{j=0}^{y-1} \left(M(i,j) - N(i,j) \right)^2$$

Střední kvadratickou odchylku budeme označovat **MSE** (*Mean Squared Error)*. **M** a **N** jsou monochromatické obrázky a **x** a **y** jsou jejich rozměry. PSNR potom spočítáme jako:

$$PSNR = 10 \times \log_{10} \left(\frac{MAX^2}{MSE} \right)$$

Kde **MAX** se spočítá podle bitové hloubky pixelu *b* jako: 2^b-1. Nejčastěji tedy 255. Pokud chceme porovnávat dva barevné obrázky, musíme je při porovnávání převést na monochromatické.

Jelikož se porovnávají vždy dva sobě odpovídající pixely, může dojít k tomu, že dva na první pohled velmi podobné obrázky budou mít velké PSNR. Například pokud se od sebe budou lišit tím, že jeden obrázek bude oproti druhému celý posunutý o několik pixelů. Tento způsob porovnávání je tedy čistě technický a neodpovídá lidskému chápání podobnosti, což ovšem v našem případě není na škodu.

Nevýhodou řazení obrázků podle PSNR je velká časová náročnost, protože pro porovnávání každých dvou obrázků se musí PSNR počítat znovu. To při velikostech obrázků v řádu milionů bodů a vzhledem k téměř kvadratické časové náročnosti řazení může znamenat problém.

4.2. SSIM

Oproti minulému způsobu porovnávání, které bylo technicky orientované, je metrika SSIM [WBS2004] navržena, aby co nejvíce odpovídala lidskému vnímání podobnosti. SSIM je zkratka z anglického *Structure SIMilarity*. Index SSIM nabývá hodnot <-1, 1>, kde 1 znamená shodné obrazy.

Tato metrika porovnávání obrázků sice bere v potaz lidské vnímání podobnosti, ovšem pro naše účely není příliš vhodná.

4.3. Histogramové porovnávání

Jedna z charakteristik obrazu je histogram [ZBSF2004]. Matematicky je histogram vektor výskytů jednotlivých barev v obraze. Jinak řečeno, hodnota histogramu *H* pro index *i* udává kolik pixelů v obraze má intenzitu *i*. Pokud není obraz složen pouze z jasových složek je histogram složen z více vektorů (např. u *RGB systému* ze 3 vektorů). Histogram tedy podává informaci o barevném složení obrázku, nikoliv však o umístění jednotlivých pixelů. Může se tedy stát, že dva vzhledově naprosto rozdílné obrázky mohou mít velmi podobný histogram. Ukázka histogramu je na obrázku 4.3.

Histogramy využívají například digitální fotoaparáty, které pomocí nich poznají špatně technicky vyfocené snímky. Například převládají-li moc malé nebo moc velké intenzity.

Seřazení obrázků pomocí porovnávání histogramů nám sice setřídí obrázky podle určitě charakteristiky, ale pro naše potřeby toto řazení také není příliš vhodné.

Obrázek 4.3. - Obrázek a jeho histogram

5. Nalezené řešení

Tato část práce se bude věnovat samotnému praktickému řešení zadání. Nejprve se zmíním o použitých algoritmech pro porovnávání a pro kompresi podobných obrázků. Následují kapitoly o použitých grafických knihovnách, bez kterých by implementace balícího algoritmu byla velmi obtížná. V poslední podkapitole je potom popis implementace a problémů, se kterými jsem se během práce setkal.

5.1. Použitý algoritmus porovnávání

Nejvhodnější algoritmus porovnávání pro naše účely by byl algoritmus využívající metriku porovnávání obrázků *PSNR* zmíněnou v minulé kapitole. Takový algoritmus by byl ovšem časově velmi náročný, vzhledem k tomu, že obrázky v současné době obsahují řádově miliony pixelů. Pokud bychom použili *výběrový algoritmus řazení*, kde vybereme první obrázek, ke kterému mezi zbývajícími hledáme nejpodobnější, a tak pokračujeme dokud nejsou seřazeny všechny obrázky, potom by pro seřazení **N** obrázků bylo zapotřebí **N * (N / 2)** porovnání. Přičemž u každého porovnání bychom museli počítat s miliony bodů. Celý výpočet by zpomalovalo nejenom velké množství pixelů každého obrázku, ale hlavně neustálé načítání ze souborů, protože uložit do paměti informace o všech pixelech většího množství (stovek nebo tisíců) obrázků je v podstatě nereálné.

Jedním z možných řešení, které jsem zvolil do svého programu já, je rozdělit obrázek na části, pro které potom můžeme vypočítat jejich konkrétní charakteristiku, tyto části uložit do datové struktury a při řazení používat namísto celých obrázků tyto struktury.

Já jsem se rozhodl rozdělit obrázky po částech velkých 32x32 pixelů a pro každou tuto část vypočítat průměrnou intenzitu pixelu a směrodatnou odchylku. Obě tyto hodnoty ukládám do proměnné typu *float*, která v paměti zabírá 4B. Počet částí obrázku je 1024x menší než jeho počet pixelů, každá část zabírá v paměti 8B. Paměťová náročnost tedy vychází 8kB na každý 1Mpx. Jinými slovy vytvoření 1000 takovýchto

informačních struktur je paměťově stejně náročné jako porovnávání dvou obrázků v plné kvalitě. Vypočítání hodnoty pro porovnávání probíhá tak, že se vypočítá hodnota *PSNR* pro průměrné intenzity pixelů a sečte se s hodnotou *PSNR* pro směrodatné odchylky.

Samotné porovnávání obrázků potom probíhá ve dvou krocích. V prvním kroku se vypočítají *informační struktury* pro všechny obrázky. Ve druhém se obrázky seřadí pomocí výše zmíněného algoritmu *výběrového řazení*, který využívá pro porovnávání obrázků jejich *informačních struktur*.

5.2. Použitý algoritmus komprese

Úkolem práce je využít podobnost obrázků a zvýšit s její pomocí kompresní poměr. Můj algoritmus *SPP* (*Similar Pictures Packager*) doslova využívá podobnosti obrázků.

Algoritmus komprese je následující:

1) Na vstupu jsou grafické soubory ve formátu *PNG*, které mohou být seřazené řadícím algoritmem popsaným výše.

2) Načteme první dva obrázky do paměti.

3) Zkusíme, jestli první obrázek zabírá ve formátu *JPEG2000* méně. Zapíšeme do výstupního souboru název obrázku, informaci v jakém formátu budeme obrázek ukládat, jeho velikost a data obrázku.

4) Vytvoříme třetí obrázek tak, že projdeme postupně všechny pixely obou obrázků a od pixelu prvního obrázku odečítáme odpovídající pixel druhého obrázku a výslednou hodnotu uložíme jako odpovídající pixel třetího obrázku.

5) Do výstupního souboru zapíšeme název ukládaného obrázku, informaci, zdali jsme uložili druhý obrázek nebo rozdílový třetí, jaký jsme použili při ukládání formát (*PNG*, *JPEG2000*), velikost ukládaného obrázku a data ukládaného obrázku. Co budeme ukládat vybereme podle toho, co je

v daném konkrétním případě nejvýhodnější.

6) Druhý obrázek přesuneme v paměti na místo prvního a jako druhý načteme ze souboru další obrázek ze vstupu a pokračujeme v algoritmu od bodu **4**. Postupně projdeme všechny obrázky ze vstupu.

Princip algoritmu je, že pokud jsou obrázky podobné (to znamená, že obsahují některé shodné oblasti se shodnými pixely), pixely v těchto oblastech budou po odečtení rovny nule. Komprimační metody formátů *PNG* i *JPEG2000* umí takové oblasti efektivněji uložit než pokud by obsahovaly pixely o různých hodnotách. Podobný princip se využívá i při kompresi videa, kde se některé snímky ukládají jako tzv. „klíčové“ snímky a u následujících snímků se ukládá pouze změna oproti předchozímu „klíčovému“ snímku.

Algoritmus dekomprese vypadá následovně:

1) Otevřeme vstupní soubor.

2) Ze vstupního souboru načteme název uloženého obrázku. V jakém je uložen formátu, jestli se jedná o rozdílový obrázek a kolik zabírá a podle těchto informací použijeme správný kodek a uložíme obrázek do paměti.

3) Pokud se jedná o rozdílový obrázek, původní obrázek získáme tak, že od předchozího obrázku odečítáme hodnoty sobě odpovídajících pixelů a získáváme tím hodnoty odpovídajících pixelů původního obrázku.

4) Algoritmus opakujeme od bodu **2** až do konce vstupního souboru.

Při rozbalování vlastně používáme jednoduché matematické pravidlo, že pokud:

$$\mathbf{A - B = C}$$

tak obrázek **B** zpětně získáme:

$$\mathbf{B = A - C}$$

V ideálním případě budou obrázky úplně shodné a do výstupního souboru se tedy uloží první obrázek a pak už jenom obrázky obsahující pouze pixely o nulové hodnotě.

V nejhorším možném případě, kdy budou mít obrázky tak málo podobnosti, že bude výhodnější ukládat je samotné místo jejich rozdílů, se do výstupního souboru uloží obrázky stejně, jak jsme je zadali na vstupu. Efektivnost algoritmu tedy záleží na vstupních datech a jejich vzájemné podobnosti. Jaká data měla při testování jaké výsledky je uvedeno v kapitole **Dosažené výsledky**.

5.3. Grafická knihovna LibGD

LibGD je *open source* grafická knihovna, která slouží pro načítání, vytváření a ukládání grafických formátů. Knihovna je napsaná hlavně pro práci s webovými stránkami, proto pracuje s formáty nejčastěji používanými na webu. Mezi podporované formáty patří: *JPEG*, *GIF* a *PNG*. Tato grafická knihovna je napsaná v jazyce C, konkrétně v jeho normalizované verzi ANSI C, takže jsem neměl při použití této knihovny žádné problémy při překladu v podobě nutnosti doinstalovávat některé další knihovny. Knihovna má na svých webových stránkách (*www.libgd.org*) velmi přehlednou dokumentaci pro uživatele i pro programátory. Knihovna je připravena pro okamžité použití a je pro uživatele velmi jednoduchá. Jediný parametr, který se používá při ukládání obrázků do souboru, je parametr kvality u formátu *JPEG*. Názvy funkcí a proměnných jsou navíc velmi vhodně zvoleny, a proto mohu vřele doporučit její používání.

K práci na svém programu jsem využil podporu formátu *PNG* konkrétně funkce pro jeho načítání a ukládání:

1) **gdImagePtr gdImageCreateFromPng(FILE *f) –** tato funkce načte soubor z obrázkem ve formátu *PNG* a obrazové informace uloží do speciální struktury *gdImage*. Jako parametr se zadává ukazatel na již otevřený soubor.
2) **void gdImagePng(gdImagePtr i, FILE *f) –** ze struktury *gdImage* uloží obrázek do souboru ve formátu *PNG*. Jako parametr se zadává struktura, kterou chceme ukládat a ukazatel na již otevřený soubor, do kterého budeme ukládat.

Obě tyto funkce jsem obalil vlastními funkcemi, abych měl pro funkce pracující s grafickými soubory sjednocené názvy. Grafická knihovna *OpenJPEG*, o které se

zmíním později, totiž pracuje s grafickými soubory trochu jiným způsobem a používá i jiný styl názvů pro své funkce.

5.4. Grafická knihovna OpenJPEG

OpenJPEG je *open source* knihovna pro ukládání a načítání obrázků ve formátu *JPEG2000* napsaná v programovacím jazyce C. Tato knihovna nemá na rozdíl od knihovny *libGD* přímo implementované funkce pro jednoduché ukládání obrázků do souborů ve formátu *JPEG2000* nebo pro jejich načítání. Tyto funkce si musí programátor napsat sám pomocí obsažených funkcí pro kódování a dekódování obrázků. Knihovna *OpenJPEG* umožňuje při kódování obrázků nastavovat velké množství parametrů, takže její použití není tak intuitivní a uživatelsky příjemné jako u minulé knihovny. Na stránkách knihovny (*www.openjpeg.org*) je programátorská dokumentace pouze vygenerovaná z komentářů ze zdrojových kódů. S touto dokumentací je tedy obtížnější práce než s dokumentací knihovny *libGD*.

Použité funkce jsou popsány v podkapitole s názvem **Implementace**.

5.5. Implementace

K implementaci jsem se rozhodl použít programovací jazyk C, který je charakteristický svojí vysokou rychlostí, a proto je vhodný pro konzolové aplikace jako je tato. Dalším důvodem bylo, že jsem se chtěl zdokonalit v programování v tomto jazyce, protože většinu studijních prací jsem realizoval v programovacím jazyku Java.

Pro uchování načtených obrázků v paměti jsem se rozhodl použít strukturu *gdImage* z knihovny *libGD*. Tato struktura udržuje informace o jednotlivých pixelech v jednorozměrném poli proměnných typu *int*, pokud se jedná o barevný obrázek s barevnou hloubkou 24b (*true color*), kde každý pixel nese přímo informaci o intenzitě jednotlivých barevných složek. V případě, že se jedná obrázek s určenou barevnou paletou (až 256 barev) nebo o šedotónový obrázek, ukládá informace o pixelech do jednorozměrného pole proměnných typu *char*. V tomto případě nese pixel informaci o pořadí barvy z použité palety. Výhoda použití této struktury je, že při vypočítávání

rozdílového obrázku stačí jednoduše procházet všechny pixely obrázku a odpovídající dvojice odečítat od sebe. Další výhodou je, že tuto strukturu používá knihovna *libGD* při načítání a ukládání obrázků ve formátu *PNG*, proto při práci s tímto formátem není zapotřebí žádná úprava této struktury. Detailní informace o této struktuře může čtenář nalézt v přehledné dokumentaci na stránkách knihovny *libGD* (*www.libgd.org*).

Pro práci ze soubory ve formátu *PNG* jsem si tedy vytvořil dvě hlavní obalující funkce:

1) **gdImagePtr imageFromPng(char *file)** – funkce načte obrázek ze souboru a uloží jej do struktury *gdImage*, jako parametr vyžaduje název vstupního souboru.
2) **int pngFromImage(gdImagePtr image)** – funkce uloží obrázek do souboru, jako parametr vyžaduje odkaz na strukturu *gdImage*.

Druhá použitá grafická knihovna *openJPEG* používá k uchování obrázku v paměti rozdílnou strukturu (*opj_image*) než knihovna *libGD*. Hlavní rozdíly jsou, že informace o pixelech ukládá do dvourozměrného pole a že pro každou barevnou složku má toto pole vytvořené zvlášť. Tedy pro barevný obrázek o třech barevných složkách (RGB) má struktura vytvořené 3 dvourozměrné pole proměnných typu *int* o velikosti šířky a výšky obrázku. Detailní informace o této struktuře jsou v dokumentaci knihovny *openJPEG* na stránkách (*www.openjpeg.org*). Pro zjednodušení práce s touto knihovnou jsem napsal dvě funkce pro převod mezi strukturami knihovny *openJPEG* a knihovny *libGD*.

Dalším problémem při použití této knihovny byla absence jednoduchých funkcí pro načítání a ukládání obrázků do souboru. Pro práce ze soubory ve formátu *JPEG2000* jsem si tedy vytvořil dvě hlavní funkce:

1) **gdImagePtr imageFromJ2k(char *file)** – funkce načte obrázek ze souboru a uloží jej do struktury *gdImage*, jako parametr vyžaduje název vstupního souboru. Ve svém těle funkce nejprve otevře soubor, vytvoří dekodér a nastaví jeho parametry, potom pomocí dekodéru vytvoří obrázek a uloží ho do struktury *opj_image*. Nakonec zavolá funkci pro převod struktury na typ *gdImage* a vrátí ukazatel na tuto vytvořenou strukturu.

2) **int j2kFromImage(gdImagePtr image)** – funkce uloží obrázek do souboru, jako parametr vyžaduje odkaz na strukturu *gdImage*. Ve svém těle nejprve funkce převede strukturu *gdImage* na strukturu *opj_image*, kterou využívá knihovna *openJPEG*. Potom vytvoří kodér a nastaví jeho parametry na neztrátovou verzi *JPEG2000*. Pomocí kodéru potom vytvoří soubor obsahující obrázek ve formátu *JPEG2000*.

V programu jsou potom pomocí výše zmíněných funkcí naimplementovány algoritmy komprese a dekomprese popsané v podkapitole s názvem **Použitý algoritmus komprese**. Nebudu zde uvádět konkrétní zdrojové kódy, na ty je možno nahlédnout na přiloženém CD.

Největším problémem, se kterým jsem se při implementaci setkal, byl *bug* při nastavování kodéru. Při použití přednastavených hodnot parametrů komprese a zároveň při použití bezztrátové komprese program vyvolal výjimku. Tento problém se mi podařilo odstranit nahlédnutím do zdrojových kódů knihovny a použitím jiných hodnot kompresních parametrů.

6. Dosažené výsledky

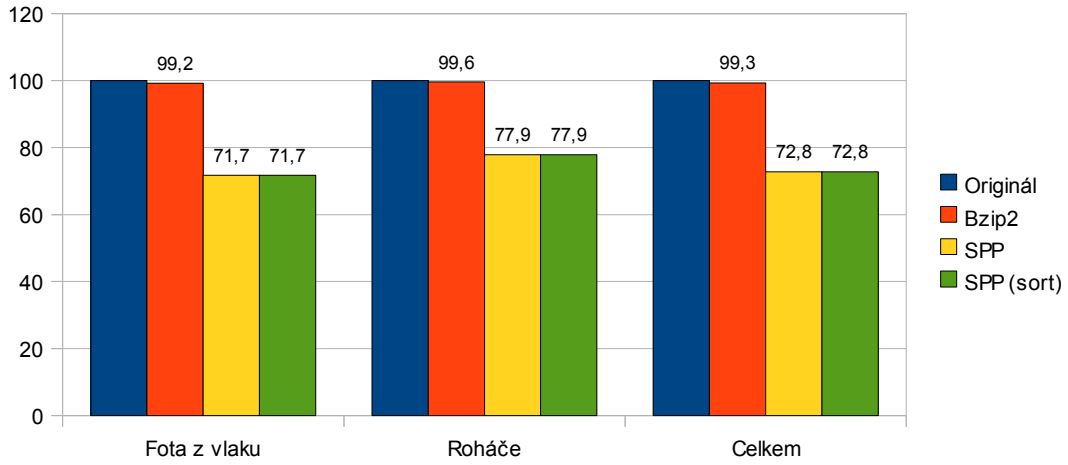
Jako vstupní data byly použity *PNG* soubory uložené v jednom adresáři. Originální velikost je tedy počítána jako součet velikostí všech těchto souborů. Je třeba mít na paměti, že ve skutečnosti takto uložená data zabírají na disku více místa než pokud jsou uložena v jednom souboru.

U všech typů dat došlo k jistému snížení velikosti a to i v případě, že podobnost mezi obrázky byla minimální či v podstatě nulová. To je způsobeno ukládáním do bezztrátového formátu *JPEG2000*, který má ve většině případů větší kompresní poměr než formát *PNG*. Nejlepších výsledků bylo dosaženo u statické filmové scény, překvapivě špatně pak dopadly snímky z CT scanu, kde bylo ovšem horších výsledků dosaženo kvůli velkému šumu obsaženého v těchto snímcích.

6.1. Náhodné fotografie

Jako první vyzkoušíme výsledný program na náhodných fotografií přírody. První sada fotografií obsahuje 19 fotek přírody focené z vlaku, druhá 5 fotek z dovolené v Roháčích. Že nedošlo ke snížení velikosti díky nějaké podobnosti je vidět i z toho, že seřazení souborů nemá na výslednou velikost žádný vliv. Viz graf 6.1.

Všechny fotografie jsou v rozlišení 3Mpx (2048x1536 pixelů) s barevnou hloubkou 24b.



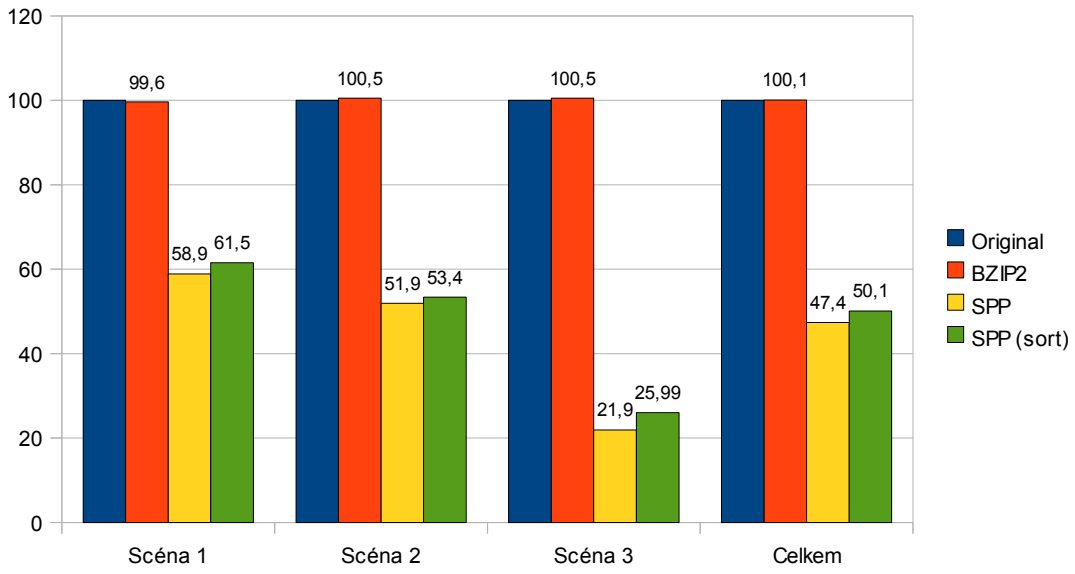
Graf 6.1. - velikost souborů v procentech

6.2. Snímky z videa

K testování byly použity snímky z animovaného filmu Elephants Dream stažené ze stránek filmu (http://www.elephantsdream.org/). Z filmu jsou náhodně vybrány 3 scény po 100 po sobě jdoucích obrázcích. U první scény se hýbe kamera, další dvě scény mají statickou kameru. Z výsledků je vidět, že při statické kameře a tudíž i víceméně neměnnému pozadí je kompresní poměr vyšší. Dále je vidět, že při použití řadící funkce se výsledná velikost souboru mírně zvýší. Viz graf 6.2.

Obrázky jsou po 100 kusech v rozlišení 640x360 pixelů o barevné hloubce 24b. Ukázky z filmy jsou na obrázku 6.2.

Obrázek 6.2. - Ukázky z filmu Elephants Dream

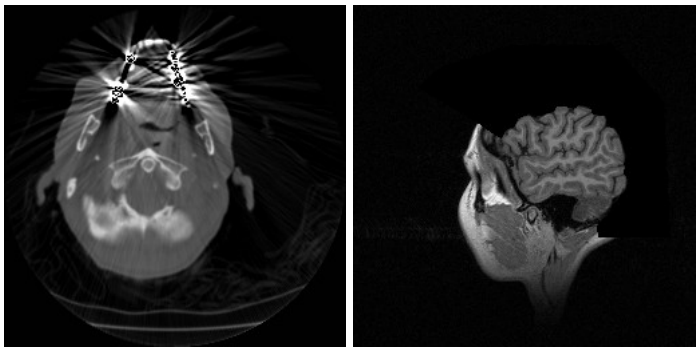


Graf 6.2. - velikost souborů v procentech

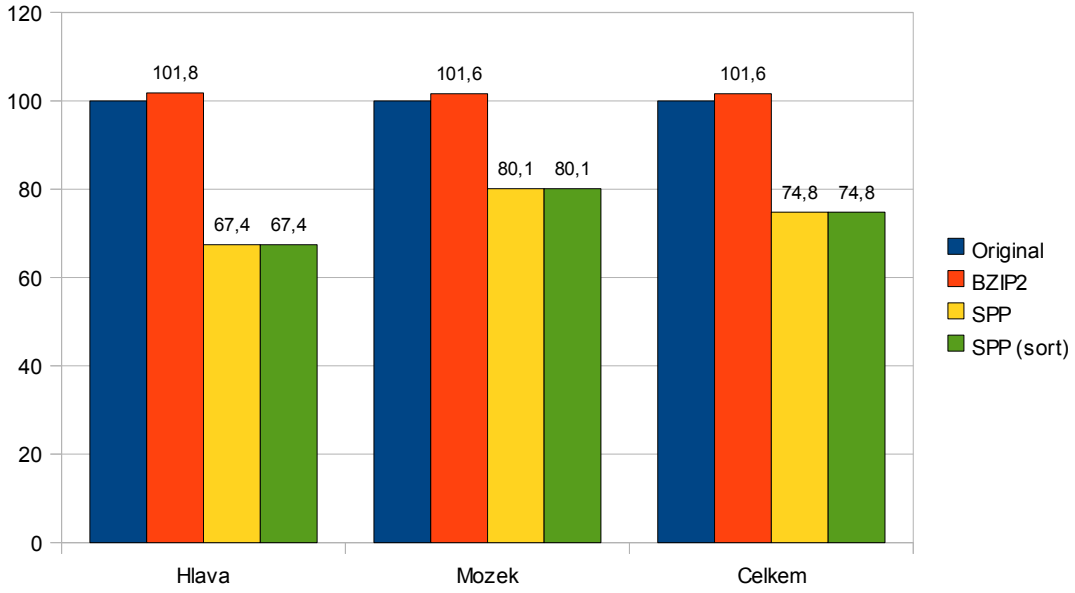
6.3. Snímky z CT scanu

K testování byly použity snímky z CT scanu lidské hlavy a lidského mozku stažené ze stránek Stanfordské univerzity (http://graphics.stanford.edu/data/voldata/). Výsledný kompresní poměr není příliš vysoký vlivem šumu (viz obrázek 6.3.), který je v obrázcích obsažen. Viz graf 6.3.

Obě sady obsahují 100 obrázků v rozlišení 256x256 pixelů o barevné hloubce 8b.



Obrázek 6.3. - Ukázky obrázků CT scanu lidské hlavy a mozku, na obrázcích je patrný zmíněný šum



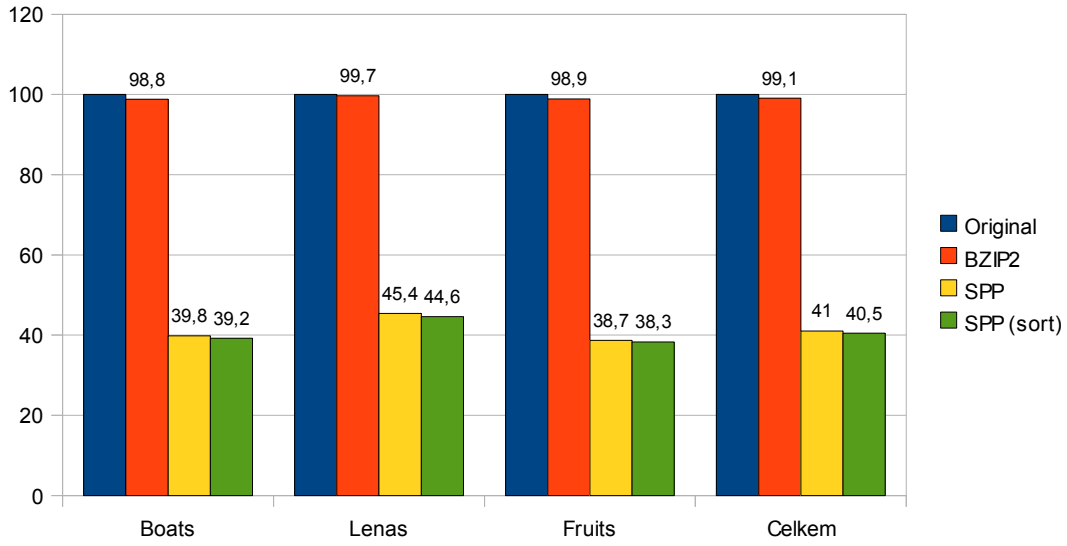
Graf 6.3. - velikost souborů v procentech

6.4. Stejné obrázky v různé kvalitě

Tato data byla dodána vedoucím práce. Jedná se o 3 sady třiceti černobílých obrázků, kde jsou jednotlivé obrázky v sadě vytvořeny postupným snižováním kvality prvního obrázku (viz obrázek 6.4.2.). U tohoto typu dat je dosaženo slušného kompresního poměru v průměru okolo 60%, který se ještě mírně zlepší použijeme-li před komprimací řadící algoritmus. Viz graf 6.4.

Všechny obrázky jsou v rozlišení 512x512 pixelů o barevné hloubce 24b. Ukázky obrázků jsou na obrázku 6.4.1.

Obrázek 6.4.1. - Ukázky obrázků zleva Boats, Lenas, Fruits



Graf 6.4. - velikost souborů v procentech

Obrázek 6.4.2. - Degradace kvality u obrázků Leny (použit každý druhý obrázek)

7. Závěr

Zadáním práce bylo nalézt podobnost ve skupině obrázků a s její pomocí co nejvíce vylepšit kompresní poměr, přičemž nesmí dojít k žádné ztrátě informace. Z výsledků dosažených při testování algoritmu je vidět, že u vhodných dat je výsledná velikost souboru výrazně nižší než pokud použijeme kompresi, která podobnosti nevyužívá.

Jako jeden z úspěchů práce považuji úspěšné použití knihovny *openJPEG* a tím pádem využití bezztrátového formátu *JPEG2000.* Díky použití tohoto formátu se snížila velikost dat zhruba o 25% i bez využití jejich podobnosti. V kombinaci využití podobnosti a komprese formátů *PNG* a *JPEG2000* se potom v nejlepším případě podařilo zmenšit data až na necelých 22% oproti původní velikosti (tedy o více než 78%). Při testování algoritmu se také ukázalo, že naprostá většina rozdílových obrázků se ukládala ve formátu *PNG*, zatímco „klíčové“ obrázky (obrázky, které se ukládaly ve stejné podobě jako na vstupu) se ukládaly v naprosté většině případů ve formátu *JPEG2000.* Použití formátu *JPEG2000* tedy pomohlo snížit velikost hlavně v případech méně podobných dat, což jsou ovšem v praxi nejčastější data.

Použití řadícího algoritmu pomohlo vylepšit výsledek jen v některých případech. Je třeba si uvědomit, že pro urychlení řazení jsou data zprůměrována a navíc to, že jsou dva obrázky podle použité metriky podobnější neznamená, že rozdílový obrázek bude po zakódování do formátu *PNG* nebo *JPEG2000* zabírat méně místa.

Literatura

[SM1996] SOBOTA Bronislav, MILIÁN Ján. *Grafické formáty*.1. vydání. České Budějovice: Grada, 1996. 157 s. ISBN 80-85828-58-8

[ZBSF2004] ŽÁRA Jiří, BENEŠ Bedřich, SOCHOR Jiří, FELKEL Petr. *Moderní počítačová grafika: kompletní průvodce metodami 2D a 3D grafiky*. 2. přepracované a rozšířené vydání. Brno: Computer Press, 2004. 609 s. ISBN 80-251-0454-0

[MR1995] MURRAY James D., VANRYPER William. *Encyklopedie grafických formátů.* 1. vydání. Praha: Computer Press, 1995. 736s. ISBN 80-85896-18-4

[WBS2004] WANG Z., BOVIK A. C., SHEIKH H. R. Image Quality Assessment: From Error Visbility to Structural Similarity. *IEEE Transactions on Image Processing.* Ročník 13, číslo 4, 2004. s. 600-612. doi: 10.1109/TIP.2003.819861

Elektronické zdroje

[O2010] OBERHUMER Markus. *oberhumer.com: LZO real-time data compression library* [online]. c2010, [citováno 3.května 2010] Dostupné z URL <http://www.oberhumer.com/opensource/lzo/>

[M2010] MAHONEY Matt. *Data Compression Programs* [online]. c2010, poslední revize 17.1.2010 [citováno 3.května 2010] Dostupné z URL <http://www.mattmahoney.net/dc/>