Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Statistické kódování digitálního obrazu

Plzeň 2011 Lukáš Svoboda

Prohlášení

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

V Plzni dne 11. května 2011

Lukáš Svoboda

Poděkování

Děkuji panu Ing. Josefu Kohoutovi, Ph.D. za konzultace, jeho cenné zkušenosti, rady a hlavně za zajímavý nápad k realizaci.

Abstract

Statistical Encoding of Images
This thesis proposes a new lossy format for storing digital images that is based on the detection of segments in images and their independent compression. For each segment in the picture, its boundary is detected and encoded using an arithmetic or Huffman encoding. An average colour of pixels in the segment is stored altogether with the standard deviation of these colours. In the decoder, segments are successively reconstructed from the stored boundaries and colours of their pixels are generated randomly in such a manner to have a Gaussian distribution with the stored mean and deviation parameters.

The proposed format was experimentally compared with popular JPEG format to decide in which situation could be our new format competitive. New format perfectly works in images which segments using less number of colors or have a simpler shapes. Our experiments also involved subjective assessment of format by a sort of people to make an appropriate conclusion. 81% of respondents agreed on the usefulness of the proposed format, especially, in case that low bit-rate is needed (such as on mobile devices, tablets etc.).

Obsah

1 Úvod

1.1 Motivace

Žijeme v době rozmachu chytrých telefonů, tabletů a jiných mobilních zařízení a všudepřítomném internetu. Bohužel rychlost, cena a kvalita mobilního internetu nejen u nás, ale i v mnoha jiných zemích se nerozvíjí tak rychle, jak bychom si přáli a potřebovali. S tím spojený fakt, že každý web se snaží vypadat co „nejmoderněji“, (bohužel slovo „moderní“ si dnes vykládá každý jinak), a tak je spousta webů přeplněna různými obrázky, reklamami atp. Na datové přenosy jsou poté kladeny vysoké nároky a doba načítání stránek se stává neúnosná.
Na základě těchto a mnoha podobných problémů byl stvořen tento projekt. Kdo z nás vlastně potřebuje mít 100% kvalitu obrázků na webech? Domnívám se, že většina lidí by tuto kvalitu na úkor rychlosti oželela.

1.2 Nový formát obrázků

Naším cílem je vytvořit nový ztrátový formát obrázků, který bude v některých ohledech konkurovat nejrozšířenějšímu formátu *JPEG*. Důraz je kladen především na velikost výsledného obrázku se zachováním rozumné informační kvality[1]. Poté bude třeba výsledky co nejobjektivněji ohodnotit prostřednictvím ankety s cílenými dotazy a snažit se výsledky srovnat s konkurenčním formátem *JPEG*.

[1] Tak, aby stále byla patrná informace obsažená v obrázku.

2 Základní teorie

Než se pustíme do podrobného popisu našeho formátu, je třeba si velmi stručně říci, jak funguje dnes nejrozšířenější formát *JPEG*[1] a dát čtenáři alespoň částečné teoretické povědomí a možnosti porovnání.

2.1 JPEG

JPEG komprese dosahuje vysokého kompresního poměru, až 1:100. První použití tohoto formátu se datuje od roku 1991 a využívá takzvanou *ztrátovou komprimační metodu*. Komprese je založena na faktu, že na malé změny barvy je lidské oko méně citlivé než na malé změny jasu. Nevýznamné změny barev jsou odstraňovány, změny jasu jsou naopak s co největší přesností zachovány. JPEG komprese se skládá z řady dílčích kroků.

Před kompresí dochází k rozdělení obrazu na submatice 8x8, pokud není šířka nebo výška obrázku dělitelná 8, dojde k doplnění řádků (resp. sloupců). Komprese se neprovádí na celý obrázek, nýbrž na jeho části představované submaticemi, ze kterých se složí výsledný obraz. Postupuje se zpravidla po řádkách ve směru z levého horního do pravého dolního rohu rastru. Cílem rozdělení obrazu je snížení ztráty informace při kompresi a dosažení vyššího kompresního poměru.

JPEG komprese kombinuje několik různých postupů pro zvýšení kompresního poměru. Využívá poměrně robustní matematický aparát představovaný diskrétní kosinovou transformací aplikovanou na bloky 8x8 pixelů.

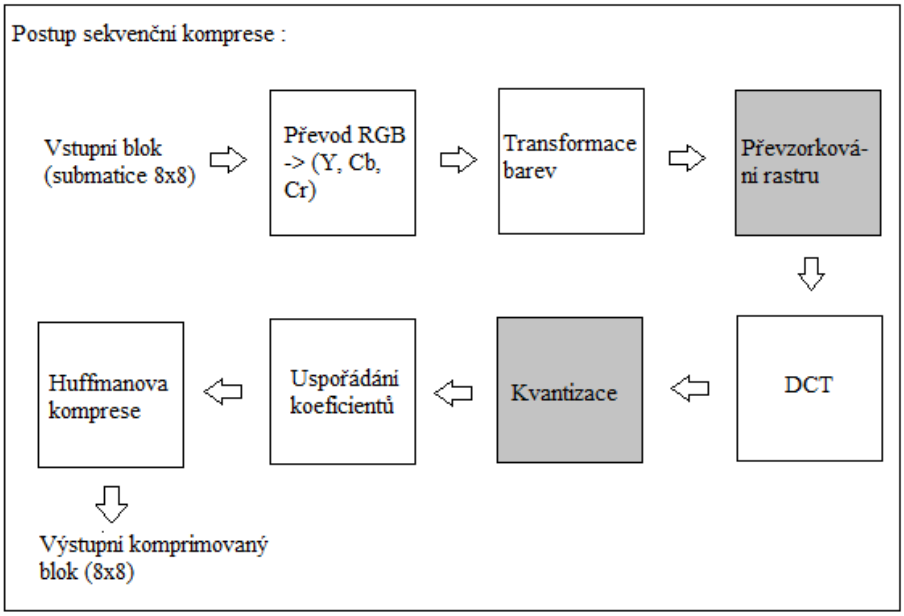
Existují 3 typy *JPEG* komprese :

- Bezztrátová komprese
- Sekvenční kódování
- Progresivní kódování

[1] Joint Photographic Experts Group.

My si zde představíme princip nejužívanějšího algoritmu sekvenční komprese (dekomprese). Více se dočtete například v literatuře [7].

2.1.1 Sekvenční kódování



Obrázek 2.1: Postup sekvenční *JPEG* komprese, šedě označené kroky jsou ztrátové.

1. **Převod dat z modelu RGB do modelu YC_BC_R.**
   JPEG využívá transformaci dat z modelu *RGB* do modelu YC_BC_R, který my rovněž používáme. *JPEG* jej používá právě proto, aby se při kompresi v podobě kosinových transformací prováděných na dané submatice co nejméně poškodila jasová složka obrázku. Protože z *RGB* modelu není možné získat přímo údaje o jasových složkách, převádí se právě do zmiňeného YC_BC_R (více v sekci 3.2 na straně 15).

2. **Transformace intervalu**
   Provádí transformaci intervalu $< 0, 255 >$ barevných složek Y, C_B, C_R na interval $<$ -255, 255 $>$, takto získané složky označíme Y', C'_B, C'_R.

   Vzorce:

$$Y' = 2 \cdot Y - 255$$

$$C'_B = 2 \cdot C_B - 255$$
$$C'_R = 2 \cdot C_R - 255$$

3. **Převzorkování (resamplování) rastru**
   Cílem je snížení počtu barevných odstínů rastrového obrázku a tím i snížení množství informací obsažených v rastru. Dosáhneme tak vyšší hodnoty kompresního poměru. Tento krok *JPEG* komprese je (jak napovídá obrázek 2.1) ztrátový. Převzorkování se nejčastěji provádí průměrováním submatic o rozměrech 2*2 pixely, 2*1 pixel (sousední pixely) nebo 3*3 (okolí pixelu).

$$f = Y = \begin{pmatrix} 192 & 172 & 168 & 172 & 165 & 175 & 178 & 148 \\ 202 & 169 & 164 & 169 & 154 & 155 & 171 & 156 \\ 210 & 162 & 151 & 160 & 147 & 124 & 137 & 150 \\ 197 & 154 & 132 & 141 & 139 & 114 & 122 & 131 \\ 206 & 174 & 137 & 129 & 138 & 113 & 113 & 113 \\ 223 & 196 & 153 & 141 & 159 & 128 & 117 & 105 \\ 192 & 227 & 189 & 153 & 163 & 138 & 130 & 93 \\ 170 & 231 & 244 & 205 & 170 & 151 & 157 & 122 \end{pmatrix}$$

Ukázka submatice f (8 x 8) představovaná složkou Y

4. **Diskrétní kosinová transformace (DCT)**
   Transformuje kódovanou oblast do frekvenční oblasti. Je bezztrátová a existuje k ní inverzní tranformace, která se používá při dekomprimaci obrazu. Vychází z předpokladu, že rastrové obrazy mají největší množství informací soustředěny v oblastech s nižšími frekvencemi, a tím umožňuje nejvýznamnější informace obsažené v obrazu zakódovat do poměrně malého množství koeficientů.

   Postup:

   - Zdrojový obraz se nejprve rozdělí na bloky 8x8 pixelů.
   - Hodnoty jasu v každém bloku se transformují z intervalu [0,2p-1] na interval [2p-1,2p-1-1]
   - Provede se diskrétní kosinová transformace následujícího vztahu.

$$\mathrm{F}(u,v) = \frac{1}{4}C(u) \cdot C(v)\Big[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y) \cdot \cos\frac{(2x+1)u\pi}{16} \cdot \cos\frac{(2y+1)u\pi}{16}\Big]$$

$$\begin{aligned} C(u), C(v) &= \frac{1}{\sqrt{2}}, \forall u, v = 0 \\ jinak : C(u), C(v) &= 1 \end{aligned}$$

Hodnota x představuje index řádkových prvků zdrojové submatice, hodnota y sloupcový index zdrojové submatice. Hodnota *f(x,y)* představuje barevnou informaci obsaženou v x-tém řádku a y-ovém sloupci. Celkem tedy u,v tvoří rádkový a sloupcový index transformované submatice, F(u,v) potom transformovanou hodnotu barevné složky (tj. prostorovou frekvenci) v tomto řádku a sloupci.

Matice F. Po provedení trasformace získáme matici, jejíž největší hodnotu má prvek s indexem *F[0][0]*, další prvky s vyššími hodnotami jsou soustředěny v levém horním rohu. Čím má prvek větší význam, tím je jeho hodnota vyšší, prvky s malou hodnotou mohou být zanedbány. Prvek F[0][0] je nazýyván DC člen, ostatní prvky AC členy.

$$\mathbf{F}(\mathbf{u}, \mathbf{v}) = \begin{pmatrix} 1270 & 125 & 20 & \dots \\ -3 & -80 & -35 & \dots \\ \vdots & \vdots & \ddots & \end{pmatrix}$$

Submatice F (u,v) po provedení DCT

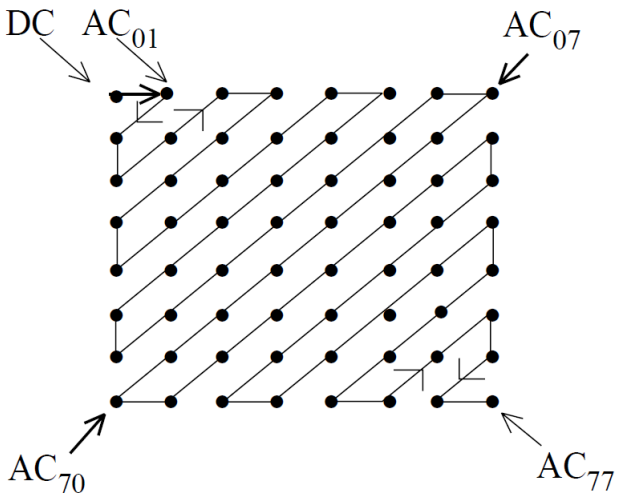
5. **Kvantizace DCT koeficientů (dekvantizace)**
Představuje nejvíce ztrátovou část *JPEG* komprese. V tomto kroku je každý z 64 koeficientů DCT vydělen (vynásoben) odpovídajícím prvkem kvantizační matice a zaokrouhlí se na nejbližší celé číslo. Cílem je vypuštění koeficientů představujících vysoké frekvence, na které není lidské oko příliš citlivé.

Kvantizace	Dekvantizace
$\mathbf{F^Q}(\mathbf{u}, \mathbf{v}) \doteq \left(\frac{F(u,v)}{Q(u,v)}\right)$	$\mathbf{F^{Q'}}(\mathbf{u}, \mathbf{v}) = F(u,v) \cdot Q(u,v)$

Vlivem možnosti rozdílného nastavení faktoru komprese q (v %), bude kvantizační matice *Q(u,v)* rozdílná pro prvky Y i C_B, C_R.

$$\begin{aligned} \mathbf{Q_q} &= \frac{Q_{50}(100 - q)}{50} \qquad q \in (50, 100) \\ \mathbf{Q_q} &= \frac{50 \cdot Q_{50}}{q} \qquad q \in (0, 50) \end{aligned}$$

6. **Uspořádání koeficientů do CIK-CAK sekvencí**
   Koeficienty uspořádáme do struktury, kterou nazýváme CIK-CAK sekvencí (viz obr.2.2).Cílem je dosažení nejvyšší účinosti komprese. Počet prvků posloupnosti odpovídá počtu prvků matice F_Q. Tato sekvence vytvoří takovou posloupnost prvků, kde vedle sebe budou umístěny prvky se stejnými hodnotami (zejména koncové prvky představované nulami). Čím je prvek posloupnosti dál, tím menší má vliv na kvalitu obrazu.



Obrázek 2.2: Prvky matice Fq uspořádané do CIK-CAK sekvencí.

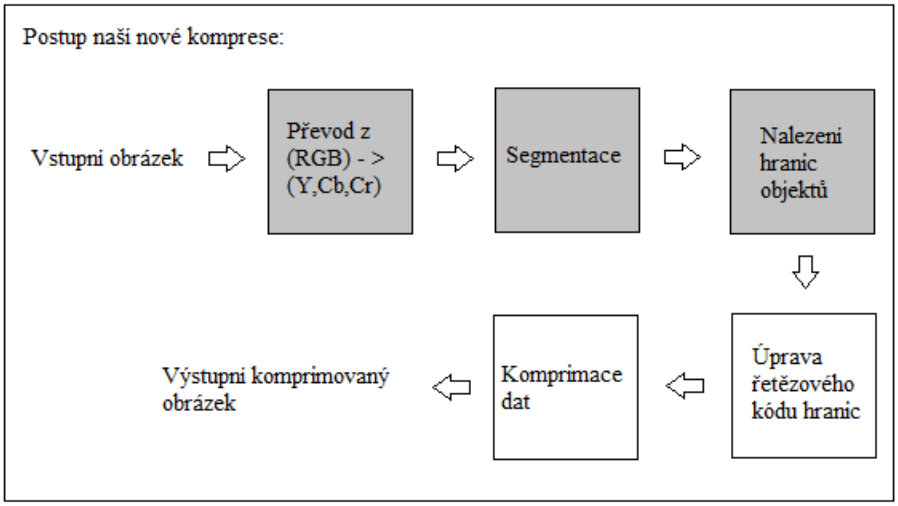
7. **Komprese** Prvky CIK-CAK sekvencí jsou komprimovány pomocí Huffmanova kódování (více v sekci 3.5.1 na straně 24).

JPEG formát je vhodný pro přirozené rastry představující fotografie, naskenované dokumenty atp. Je poměrně vhodný pro plynulé barevné přechody, avšak nevhodný pro souvislé plochy stejného barevného odstínu (kropenatost), zcela nevhodný pro černobílé rastry či technické výkresy obsahující text nebo vektorovou grafiku, u kterých se vyskytují ostré barevné přechody. V obou posledních případech dochází k takové vizuální degradaci dt, že obrázek již prakticky není možné použít. Projevuje se celkovým silným rozostřením spojeným se ztrátou hran a posunem barev. Další nevýhodou je možná ztráta informace v rastrovém souboru při opakovaném ukládání souboru.

2.2 Teoretický úvod našeho řešení

Náš formát je oproti *JPEG* principielně jednodušší. Je založen na nápadu Ing. Joseda Kohouta, Ph.D. komprimovaně ukládat jen okraje objektů (segmentů) obsažených v obrázku a průměrnou barvu, společně s průměrnou směrodatnou ochylkou od originálních barev v daných segmentech. Dodnes nebyl vytvořen žádný formát na tomto principu, proto bylo výzvou tento nápad zhotovit a srovnat funkčnost a kvalitu s ověřeným řešením.

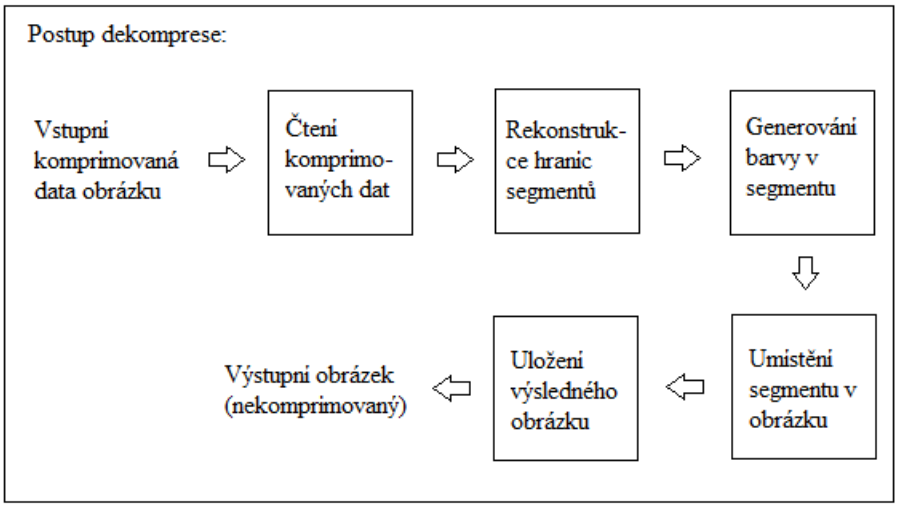
Postup komprese se skládá z několika dílčích kroků zobrazených na obrázku 2.3.



Obrázek 2.3: Postup komprese, šedě jsou vyznačeny ztrátové kroky.

2.2.1 YCbCr barevný model

Podobně jako u formátu *JPEG* i zde provádíme převod na model Y, C_B, C_R[2]. Tento převod je hojně využívaný u videa nebo u digitální fotografie. Písmeno Y zde představuje jasovou (luminiscenční) komponentu a Cb, Cr jsou modrý a červený chrominanční komponent.
Chrominanční komponent je signál pro převod barevné informace obrázku odděleně od luminiscenčního (jasového) signálu. Obvykle reprezentován jako rozdíl dvou složek : $U = B - Y; V = R - Y$. Zde Y představuje jasovou



Obrázek 2.4: Dekomprese obrázku

složku a B,R naopak modrou, respektive červenou složku.

Y, C_B, C_R není absolutní barevný model, ale jen způsob kódování RGB informací. Uplatnění má pro nás zejména díky faktu, že „zvýrazňuje“ plastičnost objektů a klade důraz na jasové přechody obrazu (viz obr. 3.2). Často také zvýrazňuje odlišnosti detailů u prolínajících se objektů, které mají podobnou barvu. Díky tomu jistě bude docházet ke kvalitnější segmentaci objektů s kvalitnějším zachováním plastičnosti a detailů obrázku (viz obr. 6.4). Více se dočtete v sekcích 3.2 a 6.1.4.

2.2.2 Segmentace

Protože potřebuji najít hranici objektů na obrázku, musím tyto objekty nějak odlišit. Pokud bych obrázek nepozměnil, hranice by se hledala mnohem obtížněji a nedokázal bych kvůli barevné odlišnosti skoro každého pixelu rozpoznat, zda se ještě jedná o daný objekt či nikoliv. Cílem je tedy objekty podobných vlastností jasně odlišit od ostatních. Zde přichází segmentace obrazu.

Segmentace[13] obrazu je jedním z nejdůležitějších úkolů automatického

zpracování obrazu. Dochází zde k patrné ztrátě kvality obrazu. Jedná se o rozdělení daného obrazu na části, kde každá část by po dokonalé segmentaci představovala jeden objekt obrazu. Spočívá ve splynutí podobných barevných segmentů do jednoho s jednotnou průměrnou barvou odpovídající průměru hodnot složek pixelů u jednotlivých spojovaných objektů (viz obr. 3.4).
Dokonalá segmentace je strojově nemožná. K její segmentaci, je-li vůbec možná, je potřeba porozumnění obsahu obrazu. Jinými slovy musíme alespoň částečně tušit, co se na obraze nachází, abychom byli schopni jednotlivé objekty v obraze přesně identifikovat a vzájemně oddělit.
Důležitým hlediskem při výběru vhodného segmentačního algoritmu je účel, pro který budeme segmentaci používat a z toho pramenící povaha vstupních obrázků. Například pro analyzování medicínských snímků z magnetické rezonance použijeme jiné algoritmy, než když analyzujeme klasické fotografie lidí nebo budov. Existuje mnoho druhů segmentací rozdělených podle metod přístupu.

Statistická metoda segmentace[4], kterou používám pracuje na principu algoritmu narůstání oblastí (*region-growing*) a jejich slučování (*merging*).
Výsledky segmentace společně s postupem řešení můžete najít v sekci 3.3 na straně 17, nebo také v sekci 6.1.2 na straně 40.

2.2.3 Nalezení hranic objektů

Poté co provedeme segmentaci, je třeba nějak popsat vzniklé oblasti. Nejpřirozenějším způsobem je popsat hranice jednotlivých objektů (oblastí) v segmentovaném obrázku.

Při segmentaci vzniká poměrně četné množství ostrých změn směru, proto jsem pro detekci hran použil řetězové kódy.
Princip řetězových kódů spočívá v tom, že cestu z jednoho bodu do sousedního bodu popisované kontury lze označit - například číslicí. Jelikož uvažujeme pohyb na úrovni jednotlivých pixelů, musíme podle zvolené provázanosti takto označit 4, respektive 8 směrů. K realizaci jsem použil 8 směrný Freemanův řetězový kód.

Freemanův řetězový kód se v oblasti úloh počítačového vidění používá pro popis tvaru objektu, který se nachází v zadaném obrázku.
Jeho osmisměrná realizace se skládá z posloupnosti čísel {*0, 1, 2, 3, 4, 5, 6, 7*}. Každé číslo představuje směr (viz obr. 3.5), kterým je třeba se posunout,

abychom přešli z jednoho bodu hranice na bod další. Detailní popis je uveden v sekci 3.4 na straně 20.

2.2.4 Úprava řetězového kódu hranic

Protože popsaná hranice objektů je v podstatě „splet'" číslic od 0 do 7, které bude třeba zkomprimovat, přemýšlelo se nad tím jak zvýšit účinnost komprese těchto číslic. Tento problém zpracovali[3] pánové Yong Kui Liu a Borut Žalik, kteří vymysleli efektivní úpravu Freemanova kódu.

Princip jejich práce spočívá v tom, že místo „klasických" Freemanových směrů se vyplatí ukládat rozdíly úhlů dvou po sobě jdoucích směrů. Zjistili průměrné pravděpodobnosti rozdílu těchto úhlů (viz tab. 2.1). Statistika byla vypracována z více jak 1000 vzorků různě tvarovaných oblastí, které byly náhodně nalezeny na internetu.

Úhel	**Pravděpodobnost**
0°	0,453
±45°	0,488
±90°	0,044
±135°	0,012
±180°	0,003

Tabulka 2.1: Statistika pravděpodobností

Na základě těchto zjištění vypracovali efektivní systém komprese pro Huffmanovo kódování, kde pro každou úhlovou diferenci (rozdíl) přísluší patřičná posloupnost bitů, která je odstupňovaná podle naměřené pravděpodobnosti (viz tab. 2.2). Více o postupu zmiňované úpravy se dozvíte v sekci 3.4.2 na straně 23.

Vlastnosti upraveného kódu:

- Osmisměrný Freemanův kód s pevnou délkou je zpravidla reprezentován 3 bity[2] pro každý směr, čtyřsměrný potřebuje pro svou realizaci 2 bity. Takto vzniklý nový kód má variabilní[3] délku, která se dá vy-

[2]Například směr 1 je kódován jako 001, směr 2 - 010 atp.

[3]Záleží na použitém obrázku a složitosti tvaru objektů - např. u „zubatých" objektů, kde dochází častěji k prudkým změnám směrů by vyšel průměrný počet bitů k reprezentaci jednoho směru vyšší, než u „hladkých" objektů.

počítat na základě naměřené statistiky jako:

$$\sum_{i=0}^{7} B_i P_i = 1 \cdot 0,453 + 2 \cdot 0,244 + 3 \cdot 0,244 \\ +4 \cdot 0,022 + 5 \cdot 0,022 + 6 \cdot 0,006 \\ +7 \cdot 0,006 + 7 \cdot 0,003 = 1,97 \textit{bitů/kód},$$

kde B_i reprezentuje směr i-tého řetězového kódu a P_i jeho naměřenou pravděpodobnost.

- K reprezentaci hranice objektů v obrázku pomocí takto upraveného řetězového kódu potřebujeme dokonce méně bitů, než při použití čtyř směrné Freemanovy růžice.

2.2.5 Komprimace dat

Nalezená data je potřeba zredukovat, avšak nesmíme o žádná data přijít. Je tedy třeba provést efektivní bezeztrátovou komprimaci. Vybrali jsme dva způsoby statistické[4] metody komprese, abychom porovnali jejich vlastnosti, praktické odlišnosti a vybrali ten nejvhodnější.

- **Huffmanovo kódování**
  Algoritmus navrhl student David Huffman v roce 1952, využívá nejkratšího prefixového kódu[5]. Metoda je založena na stanovení četnosti výskytů jednotlivých znaků v kódovaných datech a zakódování znaků s největší četností slovem s nejkratší délkou. Nejčastěji se vyskytujícím znakům je tedy přiřazen krátký kód, méně často se vyskytujícím, kód delší.

  Výhodou tohoto algoritmu bývá rychlá komprese i dekomprese a relativně malé nároky na paměť.Jeho problémem je to, že musíme znát rozdělení pravděpodobnosti výskytu jednotlivých znaků, respektive nutnost uložení binárního stromu[6] společně s daty.

[4]Ke kompresi využívá četnosti znaků vyskytujících se v datech.

[5]Prefixový kód je takový kód, jehož symboly nejsou předponou (prefixem) jiného symbolu (delšího) v kódované abecedě.

[6]Jedná se o orientovaný graf s jedním vrcholem (kořenem), z něhož existuje cesta do všech vrcholů grafu. Každý vrchol může mít maximálně dva orientované následovníky (syny) a s vyjímkou kořene právě jednoho předka.

Pro efektivnější kompresi dat reprezentujících hranice objektů využíváme již probraných znalostí (v sekci 2.2.4 na straně 10) a vycházíme z tabulky 2.2.
Samozřejmě takto upravený řetězový kód hranice má pozitivní výsledky i u jiných typů statistické komprese - například u Aritmetického kódování.

Kód	**Úhel**	**Pravděp.**	**Huffman**
C_0	0°	0,453	0
C_1	45°	0,244	10
C_2	-45°	0,244	110
C_3	90°	0,022	1110
C_4	-90°	0,022	11110
C_5	135°	0,006	111110
C_6	-135°	0,006	1111110
C_7	180°	0,003	1111111

Tabulka 2.2: Posloupnosti komprimovaných bitů

- **Aritmetické kódování**
  Aritmetické kódování[5] také patří mezi statistické metody. První kód vymyslel Peter Elias před rokem 1963, avšak praktické schéma použití vzniklo až v roce 1976.

  Na základě pravděpodobnosti každého symbolu je přiřazena odpovídající poměrná část intervalu *<0,1)*. Při kódování je pak celý interval postupně omezován z obou stran na základě postupně přicházejících symbolů. Jak se zpráva prodlužuje, zpřesňuje se i výsledný interval a jeho horní a dolní mez se k sobě přibližují. Každý symbol vybere z aktuálního intervalu odpovídající poměrnou část a ta se stane novým základem pro následující symbol.
  Kódovaná data se reprezentují libovolným reálným číslem, které leží ve výsledném intervalu získaném po přečtení všech vstupních symbolů.

3 Analýza komprese obrázku

Konstrukce (komprese) obrázku se skládá z několika dílčích kroků uvedených v sekci 2.2 na straně 7 (viz obr. 2.3). V této kapitole si podrobně popíšeme realizaci jednotlivých kroků. Než začneme s obrázkem provádět jakékoliv logické operace, je třeba obrázek nejdříve načíst do paměti.

3.1 Čtení dat

Jako základní cíl jsem si vytyčil možnost čtení obrázků v jednoduchém formátu *PPM*[1] s barevným rozsahem 8 bitů pro každou složku[2]. Tento formát jsem vybral převážně díky jednoduchému způsobu uložení (respektive načítání) dat a také proto, že u většiny zdarma dostupných prohlížečů obrázků není problém jakýkoliv obrázek do tohoto formátu převést.
Do budoucna by jistě bylo zapotřebí implementovat nejužívanější formáty obrázků dnešní doby a umožnit uživateli jednoduchý převod těchto obrázků do našeho nového formátu.

Každý obrázek se skládá z hlavičky a datové části. Hlavička specifikuje typ obrázku a jeho formát. Bez toho by prohlížeče jen těžko rozhodovaly, jaký princip mají použít při dekódování obrázku. Datová část zpravidla obsahuje samotný zakódovaný obrázek.

Pro nový obrázek vytvořím vlastní objekt typu *Picture*. Při čtení hlavičky vždy přečtu celý řádek (i včetně odřádkování „$\backslash n$“) a porovnám s referenční podobou uvedenou v předchozí tabulce. V případě neshody je program ukončen chybovým hlášením. Po přeskočení řádku s komentářem načtu rozměry obrázku a počet barev.

Po načtení hlavičky, začnu číst jednotlivé znaky (*byty*) souboru, zde každý *byte* odpovídá právě jednomu pixelu. Takto načtený obrázek si uložím do jednorozměrného pole.

[1] PPM je zkratka pro „Portable Pixel map“ a jedná se o bezztrátový formát obrázku bez komprimace.

[2] Každou složku RGB modelu představuje 256 barev.

Hlavička souboru formátu *PPM*:	
P6[LF]	znak 'P' a '6' a znak nový řádek. Specifikace formátu *PPM*[12] také povoluje místo odřádkování uvedení vlikosti obrázku a počtu barev na tomto řádku.
# comments[LF]	znak '#' a text komentáře až do znaku nového řádku a případně jejich libovolný počet na následujících řádcích.
1024 768[LF]	počet sloupců, mezera, počet řádek obrázku a odřádkování.
255	popisuje rozsah barev každé barevné složky. V tomto případě se jedná o 8 bitový obrázek (tzv. v rozsahu 0-255).
ArgbafE ...	každý pixel je představen jako jeden byte, kde hodnota 0 odpovídá bílé a 255 černé barvě. Tyto pixely jsou řazeny v obvyklém pořadí *R-G-B* (tedy červená - zelená - modrá barva).

Tabulka 3.1: Formát uložení dat v obrázku typu *PPM*

3.2 Převod z RGB modelu do YCbCr

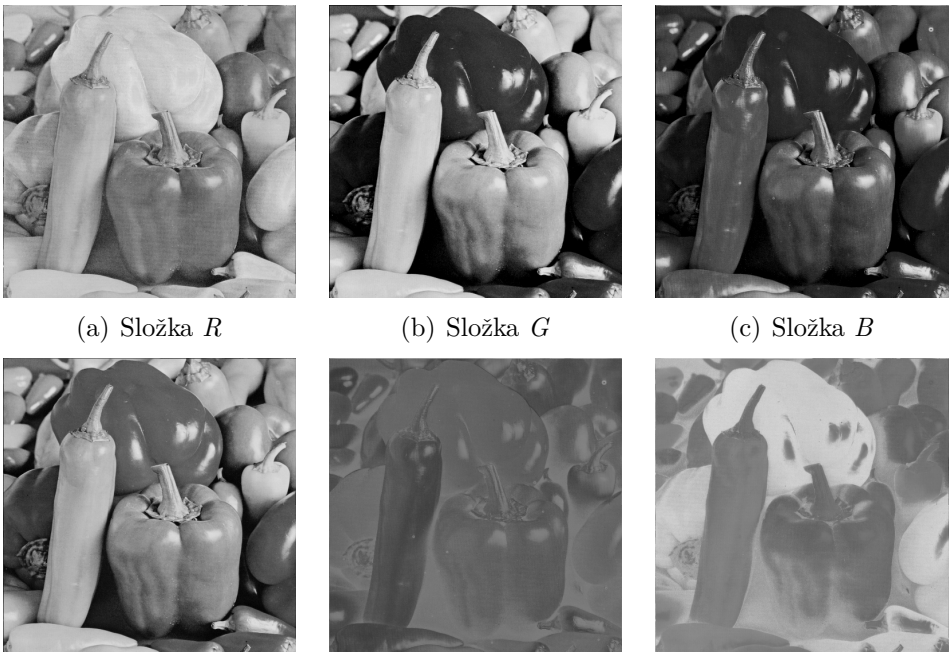
V průběhu načítání dat zároveň převádím složky *R,G,B* (viz obr. 3.1) modelu pro každý bod (pixel) do Y, C_B, C_R. K převodu využívám následující vzorce[2].

- **Převod z *RGB* do** Y, C_B, C_R

$$\begin{aligned} Y &= 16 + \frac{65,738 \cdot R}{256} + \frac{129,057 \cdot G}{256} + \frac{25,064 \cdot B}{256} \\ C_B &= 128 + \frac{-37,945 \cdot R}{256} - \frac{74,494 \cdot G}{256} + \frac{112,439 \cdot B}{256} \\ C_R &= 128 + \frac{112,439 \cdot R}{256} - \frac{94,154 \cdot G}{256} - \frac{18,285 \cdot B}{256} \end{aligned}$$

- **Pro zpětný převod z** Y, C_B, C_R **do *RGB***

$$R' = \frac{298,082 \cdot Y}{256} + \frac{408,583 \cdot C_R}{256} - 222,921$$



(a) Složka R (b) Složka G (c) Složka B

(d) Jasová složka (Y) (e) „Modrost“ (C_B) (f) „Červenost“ (C_R)

Obrázek 3.1: Obrázky jednotlivých složek modelů v šedotónové reprezentaci.

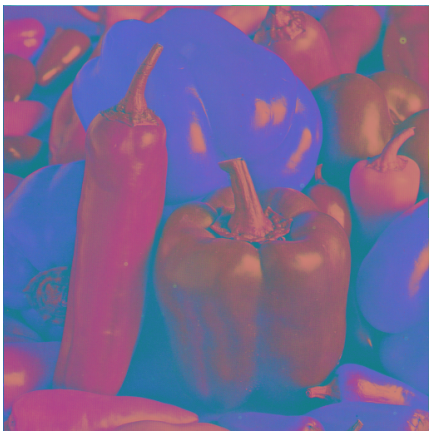
$$\begin{aligned} G' &= \frac{298,082 \cdot Y}{256} - \frac{100,291 \cdot C_B}{256} + \frac{208,120 \cdot C_R}{256} + 135,576 \\ B' &= \frac{298,082 \cdot Y}{256} + \frac{516,412 \cdot C_B}{256} - 276,836 \end{aligned}$$

Na uvedeném schéma postupu konstrukce (viz obr. 2.3) je tento krok uveden jako ztrátový. Nicméně jedná se jen o možné nepatrné ztráty vlivem zaokrouhlování na celá čísla při užívání vzorců a omezení velikosti datového typu *byte*[3].

Jak si také dále ukážeme, zmíněný převod může mít díky pravidelnějšímu tvaru ohraničení objektů u některých nastavení dokonce pozitivní dopad na velikost obrázku, avšak většinou je rozdíl velikostí takřka nulový (více v sekci 6.1.1 na straně 39).

[3]Při převodu může dojít k číslu většímu, nebo menšímu jak 0 (resp. 255) a čísla ořezávám na velikost byte.

(a) *RGB*



(b) YC_BC_R

Obrázek 3.2: Ukázka převodu do YC_BC_R.

3.3 Statistická metoda segmentace

Abych s obrázkem mohl dále pracovat, respektive najít hranice jednotlivých objektů, musím nejprve dané objekty nějak vymezit. V této kapitole si stručně popíšeme princip použité segmentace, která byla teoreticky popsána v sekci 2.2.2 na straně 8.

Obraz je postupně dělen na menší a menší oblasti do předem dané struktury a sousední oblasti se naopak zase spojují pokud splňují kritéria homogenity.

Kritérium homogenity (predikát) hraje klíčovou roli v chování algoritmu slučování segmentů (případně při jejich rozdělování). Použité kritérium by mělo brát v úvahu druh obrázku, vlastnosti objektů v obraze apod. Víme-li například, že objekty jsou tmavé a pozadí světlé, můžeme jako kritérium použít úroveň šedi v jednotlivých rozích, popřípadě úroveň šedi celé oblasti. Tvrdým oříškem pro segmentaci obrázků často bývají situace, kde se barvy jednotlivých objektů na obrázku jen nepatrně liší. Toto je demonstrováno na známém obrázku Leny (viz obr. 3.3).

Postup segmentace:

- **Vytvoření párů pixelů**
  Spárujem sousední pixely a místo těchto pixelů pracujeme s hodnotou,

Obrázek 3.3: Demonstrace slabých barevných odlišností objektů na obrázku Lena.

která přísluší maximu rozdílů barevného modelu[4] v absolutní hodnotě.

$$f(p, p') \quad = \quad max|p'_a - p_a| \qquad a \in \{R, G, B\},$$

kde p a p' jsou dva sousední pixely. Tato metoda je nazývána metodou spojovaných oblastí, výchozí poloha v podobě spojených pixelů již tvoří základní množinu elementárních oblastí, se kterou budeme dále pracovat.

- **Seřazení párů**
  U každé oblasti si zapamatujeme její pozici v obrázku a velikost. Počáteční oblasti v podobě párů pixelů vzestupně seřadíme *bucket sortem*[8] a nad takto vzniklou množinou S provádíme spojování oblastí.

- **Spojování oblastí na základě predikátu**
  Při spojování oblastí vycházím z následujících rovnic. Nejdříve si představíme rovnici, u které ovlivňujeme výslednou „kvalitu“ segmentace. Nás bude nejvíce zajímat parametr Q, který vyčísluje složitost scény, neurčitost scény a statistickou náročnost úlohy. Pokud je Q malé, model nabývá na neurčitosti ($Q = 1$ přinese nejvíce neurčitý objekt). V praxi to znamená, že Q je parametr, kterým dokážeme kontrolovat hrubost segmentace. Zpravidla bývá, že čím je Q menší, tím jsou kladeny menší nároky na toleranci odlišností spojovaných oblastí a tím je segmentace hrubší (vzniká méně objektů).
  Ještě si ukážeme vzorec ve kterém tento parametr figuruje:

[4] R, G, B, respektive YC_BC_R

$$b(R) \quad = \quad g\sqrt{\frac{1}{2Q|R|}\left(\ln\frac{|\mathcal{R}|_R}{\delta}\right)}$$

R je prozkoumávaný region a Q je nastavená konstanta „kvality“.

Jako predikát při jehož splnění spojujeme sousední oblasti z množiny S, používáme následující podmínku:

$$\mathcal{P}(R,R') \quad = \quad \begin{cases} true & \text{if } \forall a \in \{R,G,B\}, |\overline{R'}_a - \overline{R'}_a| \\ & \leq b(R) + b(R') \\ false & \text{jinak} \end{cases}$$

$\overline{R}_a$ symbolizuje pozorovanou oblast s průměrnou barvou a a $\overline{R'}_a$ je sousední segment se kterým uvažuji spojení.

- **Odstranění malých segmentů**
Protože při segmentování obrázku mohlo dojít k nepříjemnému přesegmentování obrazu, je třeba odstranit vzniklé malé segmenty. Tato metoda pracuje na principu spojování oblastí, pro kterou je nejdříve potřeba vhodně zvolit kritérium, na základě kterého budou malé segmenty připojovány ke svým sousedům.

  Toto kritérium bylo zvoleno na základě poměru malých segmentů vzhledem k velikosti obrázku. Jako malý segment byl v našem případě označen segment T_{min} menší jak 0,1% a 0,01% obrázku. Zhodnocení vlivu nastavení minimálních segmentů na kvalitu je testováno v sekci 6.1.2 na straně 40.

  Průběh algoritmu:

  1. Najdi všechny regiony s počtem pixelů menším než T_{min} a vlož je do množiny S.
  2. Procházej postupně všechny prvky a z množiny S a prováděj následující kroky, dokud S nebude prázdná.
     (a) Vymaž a z S.
     (b) Najdi pro a nejpodobnější sousední region b.

(c) Spoj a a b do nového regionu c.
(d) Vymaž b z S.
(e) Je-li c příliš malý region, zařaď jej do množiny S.

(a) Q = 32 (b) Q = 200

Obrázek 3.4: Ukázka segmentace - bez převodu na YC_BC_R model s velikostí minimálních segmentů 0,01%.

- **Převod barvy oblasti na unikátní číslo**
Abych mohl rozumně uložit data do matice (pro následnou pohodlnou detekci hran) a zároveň nepřišel o unikátní barevné zabarvení segmentu, provádím nad každým segmentem součet 3 po sobě jdoucích bytů (R-G-B) jako:

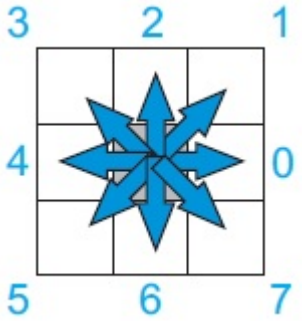
$$S_i = R_i + (G_i \cdot COL) + (B_i \cdot COL^2),$$

kde i označuje i-tý pixel (bod) segmentu, R, G, B jsou jednotlivé barevné složky každého bodu a COL je počet barev v obrázku. S je potom výsledné unikátní číslo, které uložím do matice na pozici odpovídající pozici pixelu v segmentovaném obrázku, toto provedu pro všechna i v obrázku.

3.4 Freemanův řetězový kód

Pro jednoduchost uvádím směry číslované od 0-7, nicméně ve skutečnosti používám množinu čísel {*0, 1, 2, 3, 4, -3, -2, -1*}. Proč používam takto

změněnou množinu čísel se dozvíte v sekci 3.4.2 na straně 23.

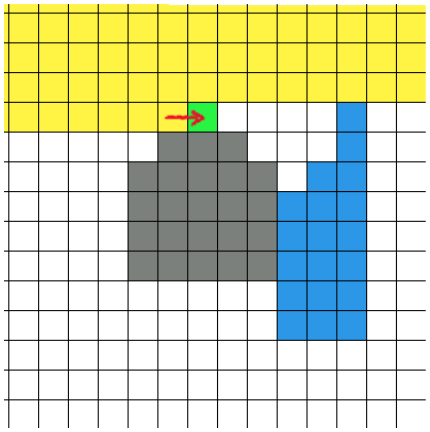


Obrázek 3.5: Směry Freemanova řetězového kódu

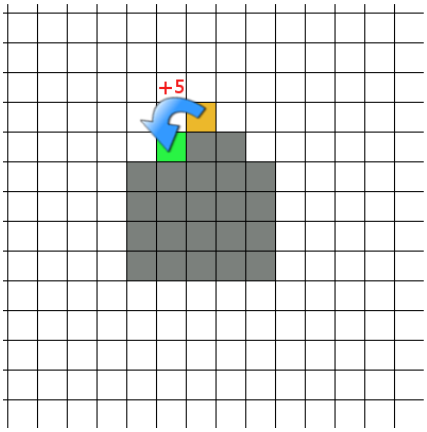
3.4.1 Algoritmus řešení

Samotný algoritmus hledání freemanova kódu vypadá následovně.

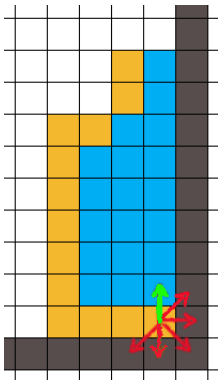
1. Projdu cyklem *for()* vstupní obrázek (resp. matici reprezentující obrázek) a hledám první bod (viz obr. 3.6) popředí. Tento bod si zapamatuji.
2. Nastavím výchozí hodnotu proměnné reprezentující směr procházení na 5.
3. K poslednímu nalezenému směru přičtu (viz obr. 3.7) hodnotu 5 (výslednou hodnotu musím dělit modulem 8).
4. Postupně funkcí procházím všechny možné směry, dokud nenaleznu další bod popředí a uložím výsledný směr do spojového seznamu objektu typu *List*. Nesmím zapomenout na ošetření hranic obrázku (viz obr. 3.8).
5. Po nalezení nového bodu popředí si tento bod zapamatuji a opakuji krok 3.
6. Kroky 3, 4 a 5 opakuji tak dlouho, dokud nedosáhnu původního bodu (viz obr. 3.9) popředí (viz krok 1). Výsledný Freemanův kód uložený ve spojovém seznamu je potřeba normalizovat.



Obrázek 3.6: Nalezení bodu popředí.



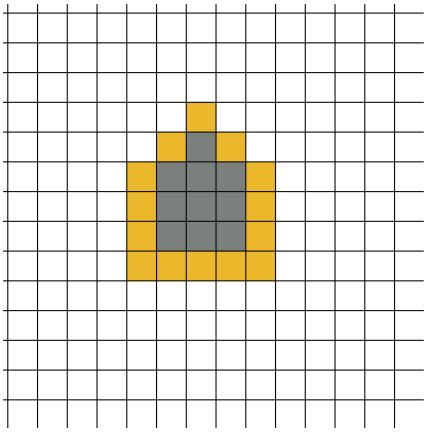
Obrázek 3.7: Přičtení hodnoty +5 k původnímu nalezenému bodu popředí.



Obrázek 3.8: Ošetření rozměrů.

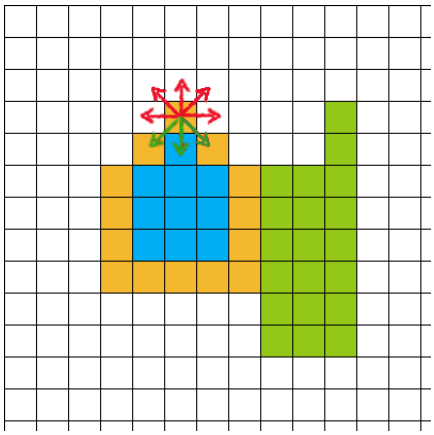
7. Pro možnost nalezení dalších objektů v obrázku, musím již prošlý objekt od ostatních nějak odlišit. Proto nad daným objektem využiji algoritmus osmisměrného semínkového plnění[6], ten jej vyplní na definovanou barvu pozadí, kterou při hledání objektů nevyužívám (viz obr. 3.11).
   V rámci tohoto kroku vypočítám v daném segmentu průměrnou směrodatnou odchylku[9] δ barev $\overline{R_{avg}}$,$\overline{G_{avg}}$,$\overline{B_{avg}}$ (respektive $Y_{avg}C_{Bavg}C_{Ravg}$) od originálních barev obrázku R_{orig},G_{orig},B_{orig}. Průměrnou odchylku pro barvu R (červenou) počítám podle následujícího vzorce :

$$\delta_R = \sqrt{\frac{1}{N-1}\left(\sum_{i=1}^{N}(R_{orig} - \overline{R_{avg}})^2\right)}$$

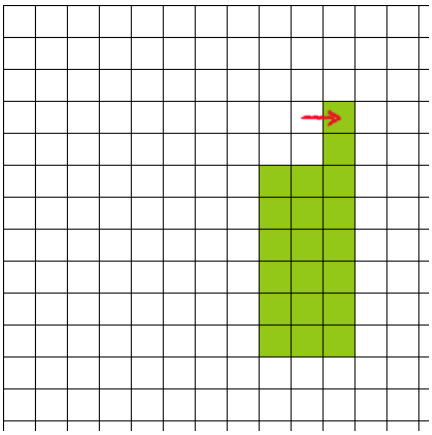


Obrázek 3.9: Výsledná nalezená hranice.

N je počet bodů (pixelů) v segmentu. Pro ostatní barvy postupuji analogicky.



Obrázek 3.10: Semínkové plnění objektu na barvu pozadí.



Obrázek 3.11: Po dokončení algoritmu semínkového plnění a nalezení dalšího objektu.

3.4.2 Úprava Freemanova kódu

Úpravu provádím na základě probrané teorie v sekci 2.2.4 na straně 10, ke zefektivnění kompresních metod.

Například místo objektu nalezeného domu (viz obr. 3.9), kde by klasická posloupnost freemanova kódu vypadala jako {5,5,6,6,6,0,0,0,0,2,2,2,3,3},

převedu na posloupnost čísel {5,0,1,0,0,2,0,0,0,2,0,0,1,0}. První číslo je vždy zachováno a ostatní čísla symbolizují úhel odchylky 2 po sobě jdoucích směrových vektorů (viz tab. 2.1). Zatímco v rámci takto upraveného řetězového kódu jsme nepřišli o žádnou informaci obsaženou v kódu původním, na první pohled jsou již patrná pozitiva v podobě většího shluku menších čísel, která mají velké uplatnění u statistických metod komprese. Před samotným převodem, v rámci hledání hranic objektů, zaměním množinu směrů Freemanovy růžice $\mathcal{Z} = \{1, 2, 3, 4, 5, 6, 7\}$ za množinu $\mathcal{Y} = \{1, 2, 3, 4, -3, -2, -1\}$.

Převod:

1. Načtu nový směr Freemanova řetězového kódu C_i ze seznamu a zapamatuji si předchozí směr C_{i-1}.
2. Dále vypočtu rozdíl těchto čísel $D_i = C_i - C_{i-1}$.
3. Pro každý rozdíl zkontroluji, zda je $D_i \leq 4$. Pokud není, mohlo D_i nabýt hodnot čísel z množiny $\mathcal{I} = \{5, 6, 7, 8\}$, nebo z množiny $\mathcal{J} = \{-4, -5, -6, -7\}$. Toto je důsledkem například odečtením záporného čísla, z upravené množiny $\mathcal{Y}$, od kladného - $D_\alpha = 4 - (-3)$ s pozicí α v množine $\mathcal{I}$ (resp. $\mathcal{J}$), kde $\alpha \in < 0, 3 >$.
   Každé takové číslo nahradím prvkem z množiny $\mathcal{K} = \{-3, -2, -1, 4\}$ (resp. $\mathcal{L} = \{-4, -5, -6, -7\}$ na pozici α.
4. Následně C_i prohlásím za D_i a kroky 1 až 3 postupně opakuji pro všechna C_i v seznamu.

3.5 Statistické metody komprese dat

Jak bylo řečeno v teoretické části, statistické metody komprese dat se využívají ke komprimaci dat pravděpodobnosti četností symbolů v kódované abecedě. Tyto metody jsou bezzstrátové, čili jsme schopni komprimovaná data opět dekomprimovat, aniž bychom o nějaká data přišli.

3.5.1 Huffmanovo kódování

Princip metody spočívá ve vytvoření binárního stromu, jehož koncové uzly odpovídají symbolům původní abecedy, hrany jsou ohodnoceny symboly 0

a 1 a uzly jsou ohodnoceny pravděpodobností výskytu. Pravděpodobnost vnitřního uzlu je přitom rovna součtu pravděpodobností jeho následníků. Uzly řadíme do posloupnosti podle rostoucí pravděpodobnosti, v každém kroku z ní odstraníme dva uzly s nejnižší prioritou, vytvoříme z nich následníky nového uzlu a ten opět zařadíme do seznamu.

Před samotnou kompresí je třeba upravený řetězový kód převést na sekvenci čísel C_i, kterým přísluší odpovídající posloupnost bitů (viz tab. 2.2).
Jako příklad uvedu upravený řetězový kód objektu domu (představen v sekci 3.4.2 na straně 23) na posloupnost čísel využívající zmíněnou tabulku 2.2 $\{5,C_0,C_1,C_0,C_0,C_2,C_0,C_0,C_0,C_2,C_0,C_0,C_1,C_0\}$.

Postup převodu:

- **Převod řetězového kódu na posloupnost bitů**
  Řekněme, že chceme převést směr *2* na posloupnost bitů. Tomuto směru odpovídá (viz tab. 2.2) sekvence bitů *1110*. Přiřadíme-li k dané sekvenci číselnou hodnotu datového typu *byte*, bude odpovídat číslu 14[5]. Nad takovým číslem provedu vhodné bitové násobení a výslednou posloupnost uložím do seznamu typu *boolean*[6], kde každému bitu *1* přísluší hodnota *true*, naopak bitu *0* hodnota *false*. Bitové násobení porovnává postupně jeden po druhém příslušné bity (první bit prvního vzoru s prvním bitem druhého vzoru, druhý bit prvního vzoru s druhým bitem druhého vzoru atd.) a provádí s každým párem logickou operaci *AND*. Hodnota bitu na dané pozici ve výsledku je 1, pokud jsou hodnoty bitů na téže pozici v obou vstupních bitových vzorech 1, jinak je hodnota bitu na dané pozici ve výsledku 0.
  K bitovému součinu upraveného řetězového kódu C_i s vyjádřenou hodnotou posloupnosti bitů (v našem případě $C_2 = 14$), používám číslo R_j s výchozí hodnotou nastavenou na *127*. Toto číslo je totiž v datovém typu byte uloženo jako posloupnost bitů *1000 0000*. Bitové násobení provádím v cyklu (*for*) celkem pro 8 hodnot čísla R_j ($j \in < 0, 7 >$) s tím, že v každém kroku provádím nad číslem R_j bitový posun vpravo. V druhém kroku bude tedy číslo R_1 vypadat jako posloupnost bitů *0100 0000*.
  Vyjde-li po bitovém součinu taková bitová posloupnost, jejíž hodnota je větší jak 0, uložím do seznamu bit *1* a pokračuji v cyklu, dokud

[5]Číslo 14 je v počítačové technice pro datový typ byte reprezentováno jako dvojkové číslo „0000 1110“.

[6]Prvky tohoto datového typu mohou nabývat pouze hodnot true, nebo false.

nenaleznu ukončovací bit *0* čísla C_i. Jediná vyjímka je číslo C_7, které je kódováno jako posloupnost sedmi bitů nastavených na *1* (viz tab. 2.2). Ukázka bitového součinu pro druhý krok cyklu (výsledná hodnota posloupnosti je rovna 0):

$$\begin{array}{rl} & 01000000 \\ AND & \underline{00001110} \\ & 00000000 \end{array}$$

- **Získání pole *bytů***
  V této fázi mám připravený seznam s posloupností bitů, které musím zkomprimovat. Jelikož Huffmanova komprese v mém případě pracuje se znaky, což jsou ve své podstatě 8 bitová čísla, je potřeba shluky 8 po sobě jdoucích bitů uložit do datových typů *byte*. Jak se dozvíme v následujícím odstavci z analýzy principu Huffmanova kódování, díky úpravě kterou jsme provedli dokážeme v ideálním případě zakódovat až 8 směrů pomocí jednoho bitu. Například pokud by se vyskytovalo v kódu častěji 8 po sobě jdoucích neměnných směrů *0*, čemuž v našem upraveném řetězovém kódu dochází poměrně často (viz tab. 2.1), výsledná hodnota *bytu* (kódovaného znaku) bude také 0 a Huffmanův kód by zvolil jako prefix (pro takovýto často se opakující se znak), který by se rovnal bitu *0* (respektive *1* - záleží na reprezentaci Huffmanova kódování).

  Princip získání pole *bytů* (shluků 8 bitů) je prostý. Do pomocného pole uložím 8 po sobě jdoucích bitů (např. {*0,1,1,1,0,0,110*}). Toto pole následně procházím v cyklu zleva, každý bit přičtu do proměnné *byte* a provedu bitový posun vlevo. Po provedení všech 8 smyček cyklu uložím vysledný „znak“ (proměnnou v datovém typu *byte*) do pole. Uváděná proměnná (kódovaného znaku) nabude hodnoty 230, a jsou v ní zakódovány směry *0,2,0,-1*. Toto provedu pro celý seznam bitů a následně data komprimuji.

Algoritmus Huffmanovy komprese:

1. **Zjištění četnosti znaků vyskytujících se v kódovaných datech.**
2. **Vytvoření binárního stromu.**
   Posloupnost jednotlivých znaků v kódovaných datech seřadíme podle četnosti (případně abecedně) postupně zleva doprava. Jednotlivé znaky

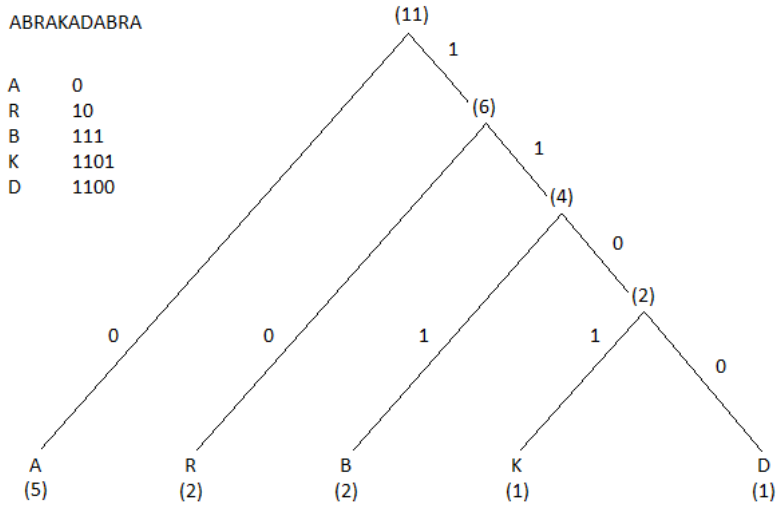
označíme za vrcholy grafu (listy stromu) a dáme je do seznamu výskytů S.

(a) Dokud je velikost seznamu alespoň dva (tzv. nacházejí-li se v seznamu stále alespoň 2 vrcholy), tak v S vybereme dva vrcholy m, n s nejmenšími počty výskytů.

(b) Vytvoříme nový vrchol, jehož levý a pravý následník (syn) je právě nalezený vrchol m (respektive n). Takto vytvořený vrchol ohodnotíme pravděpodobností výskytu, jež je rovna součtu pravděpodobností levého a pravého následníka vrcholu.

(c) Vrchol vzniklý v kroku 2b vložíme do seznamu S a vymažeme vrcholy m a n. Dále pokračujeme krokem 2a.

Následně S obsahuje kořen stromu a zbývá najít kódy jednotlivých znaků. Při průchodu z kořene do listu kódujeme 0 při kroku do levého následníka a 1 do pravého následníka (viz obr. 3.12).



Obrázek 3.12: Příklad kódování slova ABRAKADABRA.

3. **Uložení stromu.**
   Strom je uložen na začátek kódované sekvence v souboru.

4. **Nahrazení symbolů jednotlivými prefixovými kódy**, respektive posloupností bitů (viz tab. 2.2).

3.5.2 Aritmetické kódování

Algoritmus komprese:

1. **Zjištění četnosti znaků vyskytujících se v kódovaných datech.**

2. **Dělení intervalu.**
   Stanovení příslušných kumulativních (součtových) pravděpodobností *K(0)=0*, *K(i)=K(i-1) + P(i-1)* a rozdělení intervalu *<0,1)* na podintervaly I(i) odpovídající jednotlivým znakům (seřazeným podle abecedy) tak, aby délky těchto intervalů vyjadřovaly pravděpodobnosti příslušných znaků: *I(i) =< K(i), K(i+1)*.

3. **Uložení použitých pravděpodobností**

4. **Vlastní komprese**
   Začneme s intervalem *I=<0,1)*, označíme jeho dolní mez *D(I)*, horní *H(I)* a délku intervalu *L(I)=H(I) - D(I)*

   ```
   while (!eof) {
     read(i)
     I = <D(I)+K(i)*L(I), D(I)+K(i+1)*L(I))
   }
   write(D(I))
   ```

5. **Označení délky zprávy**
   Aby algoritmus rozeznal konec kódované zprávy, na začátek se přidává délka zprávy původní posloupnosti, poté následuje již samostatná kódovaná zpráva.

3.6 Uložení dat

Výsledná komprimovaná data je třeba uložit. Stejně jako každý formát obrázků, tak i náš se dělí na hlavičku a datovou část obrázku. V hlavičce jsou uvedeny základní údaje k obrázku a v datové části se nachází komprimovaná hranice oblastí. Pro porovnání rozdílů velikostí souboru (při užití 2 metod

pro kompresi hranic) je obrázek rozdělen na dvě části, kde první soubor přísluší hlavičce obrázku a druhý datové části. Toto rozdělení je do budoucna samozřejmě nepraktické a bylo by vhodné provést jednoduché sloučení hlavičky a datové části.

3.6.1 Hlavička

Soubor s hlavičkou obrázku bývá označen jako „*name_head.jkls*“, kde místo slova „*name*“ je zpravidla uveden originální název obrázku.

Hlavička souboru formátu *JKLS*[7]:

comments[LF]	text komentáře až do znaku nového řádku
Huffman[LF]	typ kódování a odřádkování
1024 768[LF]	počet sloupců, mezera, počet řádek obrázku a odřádkování
$Â:ŽţČLé ...	hlavičková data jsou vždy kódována Aritmetickým kódováním. Ukládám zde posloupnosti čísel symbolizující souřadnice počátku segmentu (x y), délku hranice (počet směrů řetězového kódu), průměrnou barvu segmentu (*R G B*) a průměrnou směrodatnou odchylku barev (*R G B*. Každé takovéto číslo je po dekódování od okolních čísel odděleno mezerou.

3.6.2 Data

V datovém souboru, označeném jako „*name_data.jkls*“ se nachází komprimovaná data směrů řetězového kódu pro všechny hranice segmentů. Data jsou uložena bez jakéhokoliv odlišení a kódována za sebou tak, jak byly nacházeny jednotlivé hranice objektů. Typ komprimace datové části je uveden také v hlavičce.

4 Dekomprese obrázku

Při rekonstrukci (dekompresi) obrázku postupuji analogicky ke krokům uvedeným v sekci 3 na straně 14 (viz obr. 2.4). Nejdříve přečtu komprimovaná data ze vstupních souborů.

4.1 Čtení komprimovaných dat

Struktura uložených dat byla probrána v patřičné sekci 3.6 na straně 28.

4.1.1 Hlavička

Při dekompresi dat zakódovaných Aritmetickým kódováním postupujeme následujícím způsobem.

1. Provedeme rekonstrukci použitých pravděpodobností.
2. Vlastní dekomprese.

```
read(X) přečteme uložené reálné číslo
while (není obnovena celá zpráva) {
  najdeme i, aby X bylo v [K(i), K(i+1))
  write(i)
  X=(X-K(i))/P(i)
}
```

Máme-li data dekomprimovaná, postupně načteme informace pro každý segment obsažený v hlavičce. Načtená data uložíme do vlastní datové struktury - vytvoříme objekty typu *Picture* a ty uložíme do seznamu *S*.

4.1.2 Data

Je-li datová část komprimována Huffmanovým kódováním, dekodér si nejprve ze souboru příslušné části obrázku načte uložený dekódovací strom, stejně

jako při kompresi, a pak zpracovává vlastní zakódovaná data.

Algoritmus dekomprese:

1. Nastaví ukazatel na vrchol stromu a pokračuje ve čtení souboru (do jeho konce).
2. Přečte další bit a posune se ve stromě na příslušnou pozici následovníka (jedná-li se o bit 0 posune se vlevo, jinak vpravo).
3. Jedná-li se o list stromu, vypíše znak reprezentovaný tímto listem a pokračuje bodem 1.

Protože jsou směry uloženy do datové struktury *byte* jako posloupnosti bitů uvedené v tabulce 2.2, je v případě Huffmanova kódování datové části potřeba po dekomprimaci dat provést dekódování jednotlivých směrů řetězového kódu.

Při dekódování postupně čteme bity dekomprimovaných znaků (*bytů*) zleva.

1. Vytvoříme pomocnou proměnnou K typu *byte*, která bude reprezentovat náš dekódovaný směr. Novému směru K přiřadíme zpočátku hodnotu 0.
2. Narazíme-li na bit *1* v dekomprimovaném znaku, přičteme daný bit ke směru K a provedeme bitový posun vlevo ve směru K. Zároveň si hlídáme počet načtených bitů.
3. Po nalezení bitu *0*, nebo je-li již počet načtených bitů v kódovaném směru K roven 7 (tzv. obsahuje posloupnost bitů *1111111*), uložíme nově vzniklý směr K a pokračujeme bodem 1.

Například při načtení *bytu* s hodnotou 2, který odpovídá posloupnosti bitů *00000010*, došlo k dekódování šesti směrů odpovídajícím hodnotě 0 (tzv. úhel 0°) a jednoho směru odpovídajícímu hodnotě 1 (úhel 45°).

Jednotlivé směry řetězového kódu jsou přiřazovány k již vytvořeným objektům typu *Picture*. Při čtení objektů z uloženého seznamu S, přečteme velikost (načtenou z hlavičky) řetězového kódu a z datové části přiřadíme danému objektu patřičnou část směrů řetězového kódu.

4.1.3 Převod upraveného kódu do Freemanovy růžice

Protože pracujeme s upraveným řetězovým kódem, tak jej před samotným algoritmem rekonstrukce převedeme do podoby „klasické“ Freemanovy růžice se směry {*1,2,3,4,-3,-2,-1*} (viz obr. 4.1).

Tento převod je vcelku prostý, sčítá vždy 2 po sobě jdoucí dekódované směry. Vyjde-li po součtu prvek D_i z množiny $\mathcal{I} = \{7, -7, 5, -5, 6, -6, -4, 8\}$, je nahrazen prvkem z množiny $\mathcal{J} = \{7, -7, 5, -5, 6, -6, -4, 8\}$ na pozici i (kde $i \in < 0, 7 >$).

4.2 Rekonstrukce hranic

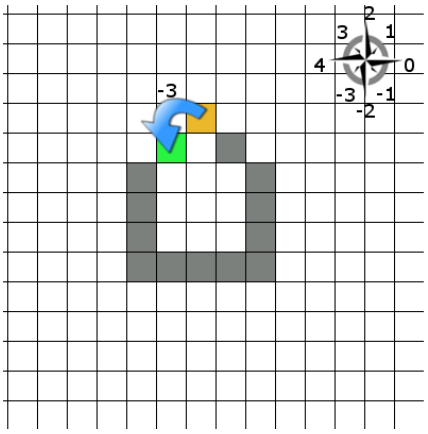
V této fázi mám pro každý segment připravena veškerá data potřebná k rekonstrukci obrázku. Nyní potřebuji každý segment vyplnit patřičnou barvou. Abych toho mohl docílit, musím nejdříve daný segment ohraničit.

Postupně procházím seznam objektů (segmentů) S a pro každý segment vytvořím pomocnou matici odpovídající celkové velikosti obrázku. U každého objektu mám v paměti uloženo, kde se nacházel první bod tohoto objektu (načteno z hlavičky) a příslušné směry Freemanova řetězového kódu od daného bodu.

1. Pro každý segment v obrázku nastavím hodnoty matice na barvu pozadí (vynuluji matici). Nastavím ukazatel na první bod segmentu a označím jej barvou hranice.
2. Postupně načítám směry Freemanova kódu a posunuji se na patřičnou pozici. Bod na nové pozici označím barvou hranice (viz obr. 4.1).
3. Bod 2 opakuji, dokud se nedostanu do výchozího bodu[1] uvedeného v kroku 1.

Nyní mohu segment vyplnit barvou.

[1] Respektive dokud jsou v seznamu stále k dispozici směry k posunutí.



Obrázek 4.1: Zeleně je označen nový nalezený bod hranice (nalezen ve směru odpovídající hodnotě -3), oranžově je značena barva již nalezené hranice a šedě zbývající hranice.

4.3 Generování barvy a umístění segmentu

Zde pracuji s průměrnou barvou segmentů $R_{avg}, G_{avg}, B_{avg}$ a směrodatnou odchylkou[9] δ_{RGB} načtenou z hlavičky. V této části rekonstrukce mám v pomocné matici na patřičném umístění odpovídajícím reálnému postavení objektu připravenou hranici a nyní je třeba daný segment vyplnit barvou.

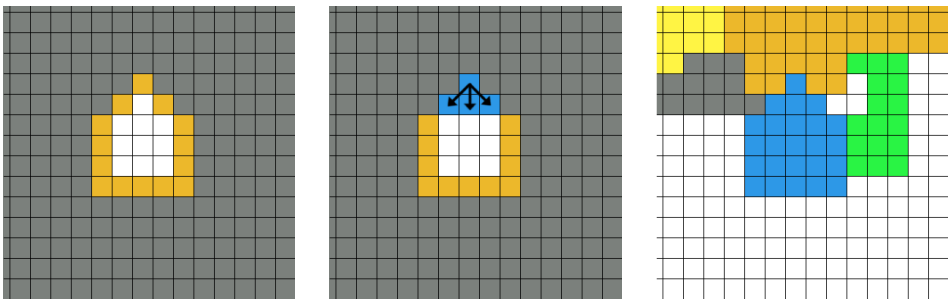
1. Z každého rohu pomocné matice provedu až k hranicím objektu semínkové plnění[6] a vyplním okolí segmentu na barvu pozadí (viz obr. 4.2(a)).

2. Poté nastavím ukazatel na první bod segmentu a od tohoto bodu provedu semínkové plnění vnitřní části[2] segmentu (viz obr. 4.2(b)). Barvu generuji náhodně v rámci funkce normálního rozdělení[9] s odchylkou δ_{RGB} od průměrných barev $R_{avg}, G_{avg}, B_{avg}$.

3. Nyní pouze umístím objekt do finální matice obrázku mezi ostatní[3] objekty (viz obr. 4.2(c)). Takzvaně zkopíruji všechny body, které neodpovídají barvě pozadí do finální matice - na stejnou pozici jako v pomocné matici.

[2]Neberu v potaz žádný bod, který je vyplněn na barvu pozadí.

[3]Mezi již zpracované objekty, které byly nalezeny před právě zpracovávaným segmentem.



(a) Nalezená hranice (b) Plnění segmentu (c) Umístění segmentu

Obrázek 4.2: Postup rekonstrukce obrázku, modře je označen zpracovávaný objekt.

4.4 Uložení rekonstruovaného obrázku

Abychom mohli vizuelně porovnat výsledný obrázek, je třeba jej uložit do nějakého formátu, který je podporován prohlížeči obrázků. Kvůli jednoduchosti jsem opět zvolil formát *PPM*[4]. Pokud jsem při komprimaci použil přechod do barevného modelu YC_BC_R, je třeba obrázek ještě před uložením převést (více v sekci 3.2 na straně 15) zpět do modelu *R,G,B*. Do hlavičky formátu uvedu pouze informace o barevné hloubce obrázku, rozlišení a poté již stačí do souboru zapsat matici výsledného obrázku v *R-G-B* modelu (viz obr. 4.3).

Obrázek 4.3: Výsledek zrekonstruovaného obrázku při kvalitě Q=500 s použitím YC_BC_R přechodu.

[4]PPM je zkratka pro „Portable Pixel map“ a jedná se o bezztrátový formát obrázku bez komprimace.

5 Implementace

Ke zhotovení práce jsem si vybral programovací jazyk *C#* 4.0. Ačkoliv jsem s jazykem před začátkem programování práce neměl vůbec žádné zkušenosti, byla pro mě jedinečná příležitost se tento moderní a stále více populární jazyk naučit. Program je rozdělen do několika souborů, ve kterých jsou části kódu rozčleněny do tříd podle vlastní datové hierarchie. V této kapitole si řekneme jen základní shrnutí používaných prostředků a jen velmi stručný popis zdrojových souborů. Na přiloženém CD naleznete kompletní programátorskou dokumentaci aplikace s popisem jednotlivých tříd, které se nalézají v níže uváděných souborech (více v Příloze C.1 na straně 64).

Struktura adresáře „*aplikace*“

- Adresář obsahuje příslušné zdrojové soubory.
  - „*ArithCoder.cs*“
  - „*Huffman.cs*“
  - „*Freeman.cs*“
  - „*IBinaryStats.cs*“
  - „*GaussGenerator.cs*“
  - „*Picture.cs*“
  - „*Point.cs*“
  - „*Program.cs*“
- Adresář „*bin*“ s podadresářem „*Release*“ obsahuje přeložený program „*Bakalářka.exe*“.

5.1 Datová vrstva

Do této vrstvy patří soubory „*Point.cs*“ a „*Picture.cs*“. Tyto soubory zajišťují zpracování dat ze zdrojového souboru obrázku a jejich následné uložení do pole typu *byte*, které využívam kvůli rychlejšímu přístupu (oproti matici) a menším paměťovým nárokům. Každému obrázku (resp. segmentu) a směrům řetězového kódu přísluší právě jeden objekt typu *Picture*.

K ukládání seznamů zpracovávaných dat využívám kolekce přístupné v knihovnách jazyka C#. Například řetězové kódy ukládám do kolekce spojového seznamu (*LinkedList*)...

5.2 Logická vrstva

Logické vrstvě přísluší soubory:

- *„Freeman.cs“*,
- *„ArithCoder.cs“* společně s rozhraním *„IBinaryStats.cs“* (rozhraní implementující třída *ArithCoder*),
- *„Segmentation.cs“*,
- *„Huffman.cs“*
- *„MyPicFormat.cs“*.

Segmentace
K segmentování obrázku jsem použil knihovnu[4] od Franka Nielsena a Richarda Nocka. Tato knihovna byla originálně vytvořena pro programovací jazyk *C++*, proto ji bylo třeba poupravit a přepsat pro naše účely. Knihovna je děděna od třídy *Picture* a provádí segmentaci obrázku. Jako datové úložiště obrázků a ostatních pomocných dat[1] je opět využíváno jednorozměrné pole. Pro seřazení párů pixelů používám *bucket sort* pro jeho snadné implementaci a složitost O(n · k). Metodu statistické segmentace jsem vybral z důvodu hotové implementace a výborných výsledků.

Freemanův algoritmus
Příslušný soubor *„Freeman.cs“* obsahuje metody ke zpracování hranice a nalezení všech objektů v obrázku. K realizaci semínkového plnění (Floodfill[6]) je použita kolekce zásobníku.

Pro implementaci Freemanova algoritmu jsem použil *switch()* v kombinaci s *for()* cyklem. Tento způsob mi přišel nejrozumnější z hlediska náročnosti zpracování i rychlosti algoritmu.

[1] Např. seznam párovaných segmentů, údaje o velikostech segmentů, jejich poloze atd.

```
switch (direction){
        case 0:
              /* Ošetření hranic objektu */
              if ((use_pixel_x + 1) < pic.getWidth() && ...
              {
                  use_pixel_x++;
                  addToList(direction);
                  break;
              }else
              {
                   direction++; break;
              }
          case 1:
               if ...
```

Huffman
K implementaci samotného algoritmu komprese jsem použil již hotovou knihovnu, kterou vytvořil Eric Nusbaum dne 22.5.2009. Tato knihovna se nalézá v souboru *Huffman.cs*. Je zde i vlastní třída *Dictionary*, která se stará o předzpracování dat (více v sekci 2.2.4 na straně 10 a sekci 3.4.2 na straně 23). Knihovnu zaobaluje třída *Huffman*, která obsahuje všechny funkční metody ke kódování dat.

Aritmetické kódování
Funkční část tohoto kódování se nalézá v souboru *„ArithCoder.cs“*. Využívám knihovnu vytvořenou panem Ing. Josefem Kohoutem Ph. D.

Konstrukce a rekonstrukce obrázku
Vlastní logická část zajišťující průběh uložení komprimovaných dat a dekompresi obrázku se nachází v souboru *„MyPicFormat.cs“*.

5.3 Hlavní program

Funkce volaná při spuštění programu se nacházi v souboru *„Program.cs“*, která zajistí běh programu a jeho výstup. Třída vytváří objekt typu *Segmentation* a *Freeman*. Dále také ošetřuje zadané argumenty pro spuštění aplikace.

6 Dosažené výsledky a testování

Aplikace byla testována prostřednictvím webové ankety, která byla vytvořena pro účely testování na adrese `http://home.zcu.cz/~svobikl`.Webová anketa byla pro účely testování vyvinuta ve spolupráci s kolegou Jaromírem Staňkem. Do této ankety byla zahrnuta množina 12 obrázků ve formátu *PNG*[1] se specifickými vlastnostmi k co možná nejobjektivnějšímu zhodnocení kvality. Byly zde velmi plastické obrázky se členitým povrchem, s vysokým počtem rozdílných zabarvení částí objektů, ale také obrázky se souvislými barevnými plochami s málo členitým povrchem (viz obr B.1). Anketu hodnotilo celkem 42 lidí, přičemž v každé anketě se stejné testy několikrát opakovaly[2] s náhodně vybraným obrázkem z množiny.

Pro pohodlné a rychlé přečtení výsledků ankety se podívejte na seznam grafů (některé jsou uvedeny i v této kapitole) v podobě obrázků (více v Příloze B.3 na straně 61).

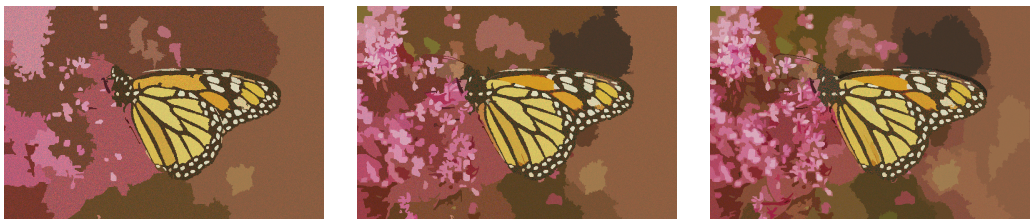
6.1 Zhodnocení různých nastavení kvality

Testována byla mezi sebou 4 různá nastavení kvality segmentace (Q = 32, 200, 500 a 1000).

V prvním formuláři ankety jsem vyobrazil vedle sebe vždy 2 obrázky a ptal se na ten, který se lidem zdál kvalitnější. Testoval jsem s využitím přechodu do YC_BC_R modelu. Ze 126 odpovědí jich 115 bylo „správně“ (tzn. označený obrázek vždy odpovídal obrázku s vyšším nastavením kvality). Také jsem se ptal, zda i na méně kvalitním obrázku je patrná informace v něm obsažená. Zde bylo celkem 101 kladných odpovědí, 23 záporných odpovědí patřilo převážně obrázkům s nejnižším nastavením kvality segmentace. K těmto odpovědím docházelo převážně z důvodu „slévání“ většího počtu okolních segmentů do sebe, kde se již u obrázků mohou ztrácet některé objekty. Avšak stále lze závěrem první fáze testování říci, že i při nejmenším nastavení kvality byla většinou informace obsažená v obrázku patrná, ačkoliv změna nastavení kvality má svá opodstatnění a je dobře rozeznatelná.

[1] PNG je formát určený pro bezeztrátovou kompresi rastrové grafiky. Znamená „Portable Network Graphics“ - tzv. přenosná síťová grafika.

[2] Pro zefektivnění výsledků hodnocení.



(a) Q = 32 (b) Q = 200 (c) Q = 500

Obrázek 6.1: Zhodnocení nastavení kvality.

Následující formulář sloužil k porovnání dvou nejvyšších nastavení kvality segmentace (Q = 500, Q = 1000). Zde jsem se snažil dokázat již zbytečně vysoké nastavení konstanty při Q = 1000, kde doba výpočtu obrázku a jeho rozdílná velikost (nárust až +30%) neodpovídá nárustu kvality. Opět jsem tedy zobrazil 2 obrázky a ptal se na kvalitnější z nich. Moje předpoklady se naplnily a u této otázky opět ze 126 odpovědí došlo k rozeznání „kvalitnějšího“ obrázku už pouze u 45 odpovědí, u dalších 13 byla uvedena odpověď „nevím“, respektive nedokázali rozpoznat rozdíl (viz obr. 6.2).

(a) Q = 500 (b) Q = 1000

Obrázek 6.2: Vliv 2 nejvyšších testovaných nastavení na výslednou kvalitu.

6.1.1 Kvalita, typ komprese a vliv na velikost obrázku

Provedl jsem srovnání velikostí u testovaných nastavení kvality segmentace. Výpočet byl proveden jako součet velikostí obrázků všech testovaných obrázků

v jednotlivých složkách napříč všemi nastaveními a porovnání těchto velikostí mezi sebou. Jako výchozí hodnotu stanovím nastavení při Q=200 s provedeným převodem na YC_BC_R barevný model (viz tab. 6.1), k testování bylo použito aritmetické kódování.

Minimální velikost segmentu = 0,01%			
Q32	Q200	Q500	Q1000
40%	102%	149%	187%
Minimální velikost segmentu = 0,01%			
Q32Y	**Q200Y**	Q500Y	Q1000Y
40%	**100%**	149%	188%
Minimální velikost segmentu = 0,1%			
Q32	Q200	Q500	Q1000
31%	65%	86%	99%
Minimální velikost segmentu = 0,1%			
Q32Y	Q200Y	Q500Y	Q1000Y
32%	67%	86%	99%

Tabulka 6.1: Procentuální srovnání vlivu nastavení kvality na výslednou velikost souboru obrázku. Označení „Y“ u testovaných kvalit znamená využití převodu do YC_RC_R barevného modelu.

Dále jsem také provedl vzájemné srovnávání vlastností u dvou používaných typů kompresních metod. Při nastavení menší kvality se projevuje nutnost ukládat výsledný strom a četnosti znaků u Huffmanova kódování, a při nastavení Q=200 je průměrný nárust velikostí souborů komprimovaných obrázků oproti Aritmetickému kódování o celých +27%.
Při nastavení Q=500% se již na výsledné velikosti tolik neprojevuje nutnost uložení stromu a nárust se tedy trochu smazává, nicméně stále je +16% v neprospěch Huffmanova kódování. Z tohoto důvodu bylo jako defaultní nastavení aplikace vybráno právě aritmetické kódování.

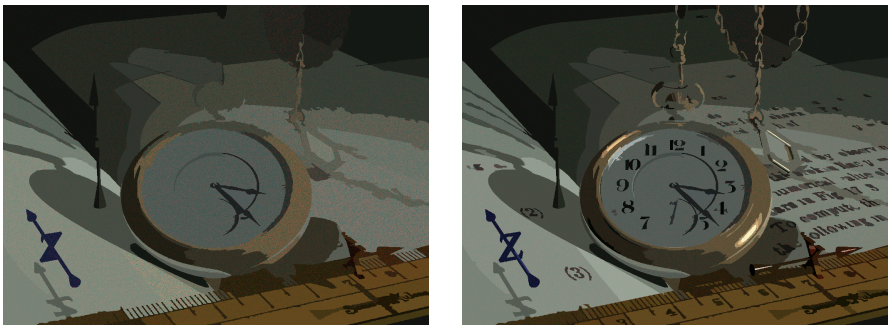
6.1.2 Nastavení minimální velikosti segmentů

Jak bylo řečeno v sekci 3.3 na straně 17, na úroveň detailů v obrázku má vliv nastavení minimální povolené oblasti segmentů.
Minimální velikost vzniklých segmentů ovlivňuje nejen zmíněnou kvalitu,

nýbrž i velikost výsledného obrázku - čím více segmentů, tím narůstá velikost obrázku (viz tab. 6.1). Testování bylo provedeno opět prostřednictvím webové ankety v kvalitě Q=500 a Q=200 u dvou nastavení minimálních segmentů, a to sice na 0,01% a 0,1% (vztaženo k celkovému počtu bodů v obrázku).

V rámci kvality Q=500 bylo nasbíráno celkem 84 odpovědí. Z celkového počtu byla drtivá většina 83 odpovědí označena „správně“. Lidé zaznamenali podstatnou změnu kvality u zobrazovaných obrázků a takřka se 100% přesností označili vždy obrázek s nastavenou velikostí minimálního segmentu na 0,01%. Také 69 odpovědí se shodovalo ohledně rozdílu kvalit dvou zobrazovaných obrázků. Rozdíl obrázků je patrný v úrovni (resp. úbytku) detailů. Jiné časté odpovědi se shodovaly v tom, že rozdíl je patrný také na barevném podání obrázku a plastičnosti objektů v obrázku. Při testování s nastavením kvality na Q=200 (viz obr. 6.3) byl výsledek velmi podobný a z 42 hlasů bylo „správně“ 36.

Díky těmto výsledkům byla při kódování obrázku do našeho formátu v aplikaci nastavena výchozí hodnota minimálních segmentů právě na 0,01%.



(a) (b)

Obrázek 6.3: Rozdíl nastavení minimálních segmentů. Na 6.3(a) je obrázek s minimálním segmentem na 0,1%, zatímco na obrázku 6.3(b) na 0,01%

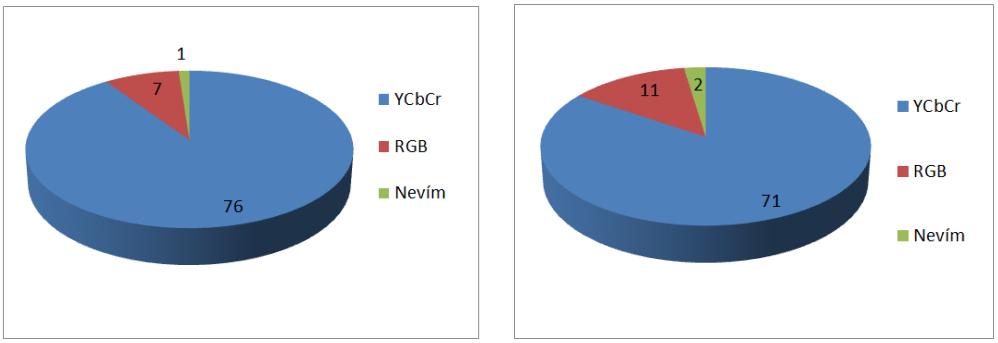
6.1.3 Vliv převodu RGB - YCbCr na kvalitu a velikost souboru

YC_RC_R využívá i hojně používaný formát JPEG a má podstatný vliv na výsledek segmentace obrázků. Společně s nulovými dopady na velikost souboru bylo využití tohoto převodu v aplikaci standartně přednastaveno.

(a) (b)

Obrázek 6.4: Rozdíl mezi RGB*(a)* barevným modelem a YC_BC_R na obrázku *(b)*. Kvalita u obrázků je nastavena na Q=500.

Na předem jasném závěru se v anketě v rámci nastavení Q=500 shodlo z počtu 84 celkem 76 odpovědí (viz obr. 6.9(a)) a v rámci 2 vyobrazených obrázků označilo jako kvalitnější ten, u kterého byl proveden převod do YC_BC_R barevného modelu. Také 56 odpovědí se na otázku ohledně rozdílu kvalit shodovalo, že dochází k výraznému rozdílu kvalit mezi těmito obrázky (viz obr. 6.4).
Při nastavení kvality na Q=200 (viz obr. 6.9(b)) byl označen jako kvalitnější obrázek s využitím YC_BC_R modelu celkem v 71 odpovědích.



(a) (b)

Obrázek 6.5: Grafy výsledků odpovědí v rámci srovnání kvality YC_BC_R a RGB barevného modelu.

6.1.4 Vliv použití posterizace na kvalitu

Efekt posterizace se používá ke zmenšení počtu barevné hloubky v daném obrázku. Testoval jsem 2 nastavení a to se stažením barevné hloubky na hodnotu, kdy se mi zdálo, že proběhne lepší segmentace. Úroveň posterizace jsem tedy nastavil na 7 a 15 (tzv. počet barev v obrázku je roven 2^7 a 2^{15}).

(a) (b) (c)

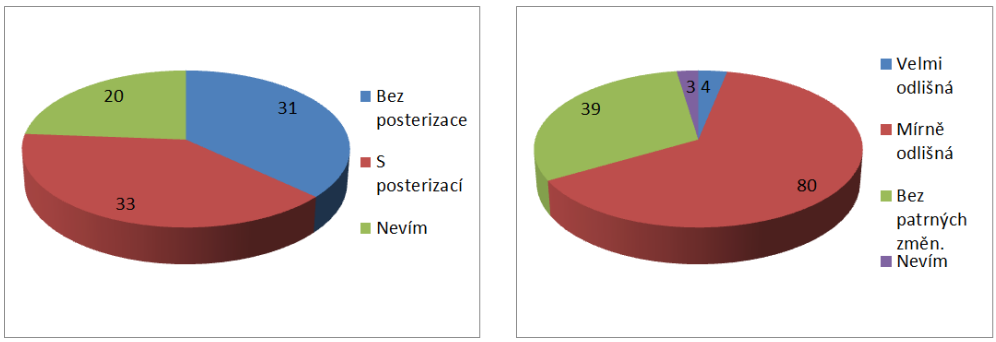
Obrázek 6.6: Obrázek *(a)* je original,*(b)* s nastavením posterizace na 7 a *(c)* s úrovní posterizace 15.

Ačkoliv byla domněnka, že u některých obrázků může vést ke zvýraznění rozdílných objektů a následné „kvalitnější“ segmentaci, reálný výsledek je ale sporný (viz obr. 6.7). Pro „dokonalé“ otestování vlivu posterizace by bylo zapotřebí provést více testování. Reálná aplikace posterizace by byla, z důvodu velmi obtížné implementace strojového rozpoznání ideálního počtu barev pro daný obrázek a jeho segmentaci, velmi obtížná. Zhodnocení nechám v tomto případě na čtenáři.

6.2 Rozdíly mezi JPEG a naším formátem

Zde demonstruji sílu našeho formátu a předvádím rozdíl mezi naším formátem a klasickým JPEG při snížené kvalitě (pro dosažení shodné velikosti obrázku). Při ukládání obrázků v JPEG jsem zvolil takovou kvalitu, abych se velikostí souboru přiblížil mému formátu. Velký rozdíl je patrný zejména u obrázků, ve kterých jsou objekty s málo členitým povrchem pokrytým souvislou barvou.

V rámci testování jsou výsledky mírně na straně formátu *JPEG* (viz



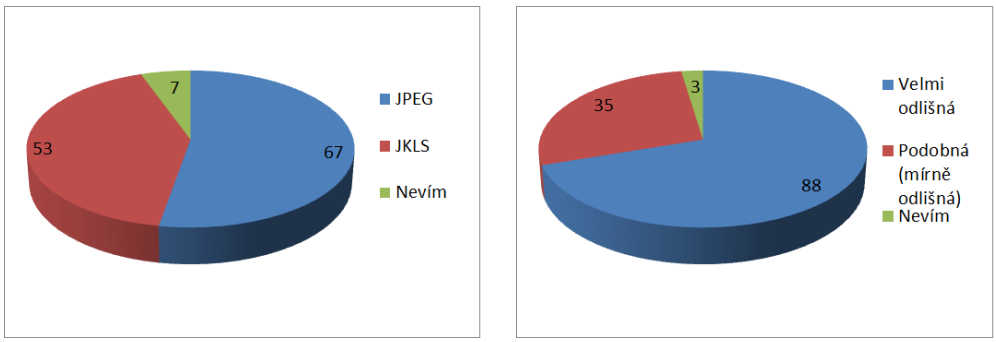
(a) (b)

Obrázek 6.7: Výsledky dotazů na zhodnocení rozdílů kvality *(b)* dvou zobrazených obrázků a také na označení kvalitnějšího obrázku *(b)*.

(a) (b)

Obrázek 6.8: Rozdíl mezi naším formátem na obrázku *(a)* a *JPEG* na obrázku *(b)* při stejné velikosti (oba 6,3kB).

obr. 6.9). Zde velmi záleží na vlastnostech objektů u použitého obrázku. U některých obrázků je komprese na straně našeho formátu (viz obr. 6.8), jinde naopak vyhrává *JPEG*. Miska vah by se dala obrátit v náš prospěch s možnými vylepšeními našeho formátu, například v podobě další až 50% úspory velikosti obrázku (více v sekci 7 na straně 47).



(a) (b)

Obrázek 6.9: Graf vyhodnocení výsledků nad dotazem pro výběr kvalitnějšího obrázku *(a)* a zhodnocení rozdílů kvalit *(b)*.

6.3 Porovnání velikostí a doba běhu aplikace

Následující tabulky (6.2 a 6.3) uvádí doby běhu programu a velikosti souborů. Časy jsou měřeny strojově od spuštění běhu programu do doby vykonání všech instrukcí (do ukončení běhu). Protože velmi záleží na typu obrázku, kde obrázky s menším počtem objektů trvají kratší dobu a zabírají méně, zvolil jsem obrázky (vyobrazeny na obr. 6.8(a),3.3 a 4.3). Hodnoty uváděné v tabulkách jsou počítány jako průměr hodnot těchto 3 obrázků. Ve všech případech testování využívám již standardně přednastavený přechod do YC_BC_R modelu.

Minimální segment	Kvalita	Náš formát (průměr).
0,01%	Q32	2,66kB
0,01%	Q200	6,66kB
0,01%	Q500	13kB
0,1%	Q32	2,33kB
0,1%	Q200	4,66kB
0,1%	Q500	7kB

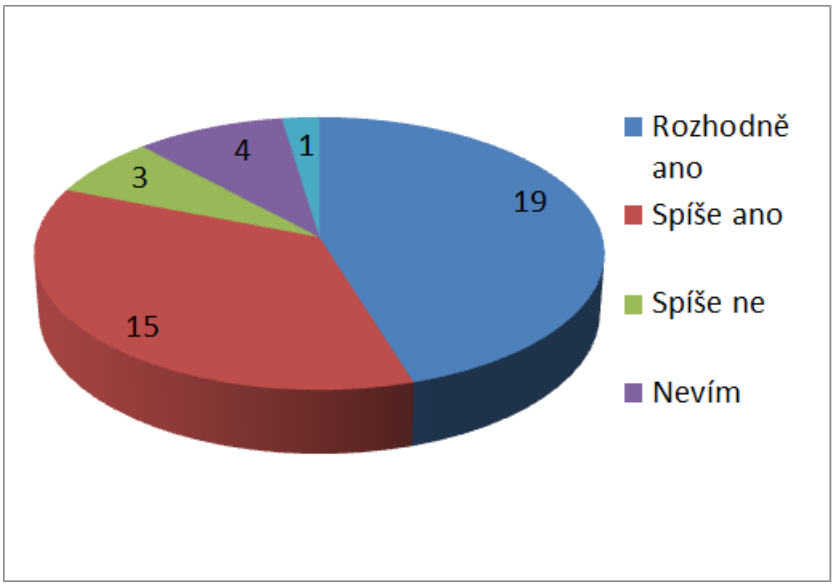
Tabulka 6.2: Srovnání velikostí souborů vůči *JPEG* ve standardní kvalitě (q=85%), u kterého je průměr souborů testovaných obrázků roven **48kB**.

		Doby běhu při komprimaci.[s:ms]	
Kvalita	Kódování	Min. segm. = 0,01%	Min. segm. = 0,1%
Q32	Aritm.	0:857	0:852
Q200	Aritm.	0:854	0:852
Q500	Aritm.	0:842	0:836
Q32	Huff.	0:904	0:904
Q200	Huff.	0:904	0:900
Q500	Huff.	0:914	0:889
		Doby běhu při dekompresi.[s:ms]	
Kvalita	Kódování	Min. segm. = 0,01%	Min. segm. = 0,1%
Q32	Aritm.	1:434	0:935
Q200	Aritm.	6:198	2:532
Q500	Aritm.	10:727	3:998
Q32	Huff.	0:904	0:904
Q200	Huff.	6:254	2:552
Q500	Huff.	10:804	3:982

Tabulka 6.3: Průměrné doby běhu programu pro 3 testované obrázky. Testováno na Core2 Duo 3.2Ghz, 4GB DDR2 Ram a Windows 7 SP1.

7 Možnosti rozšíření a použití

V poslední otázce ankety jsem se ptal, zda si lidé myslí, že má náš formát uplatnění v reálném použití (viz obr. 7.1).



Obrázek 7.1: Graf odpovědí na otázku ohledně reálného uplatnění tohoto formátu.

7.1 Příklady užití

Použití u situací, kde klasický *JPEG* nedosahuje dostatečných kvalit vzhledem k výslednému obrazu a velikosti. Síla tohoto formátu spočívá hlavně v obrázcích, ve kterých se vyskytuje méně plastických scén a u objektů se souvislými plochami barev (viz obr. 6.5). Dále se mohou dobře uplatnit na některých webových prezentacích, kde obrázky nehrají podstatnou informativní roli, nebo jako komprimaci všudepřítomných reklamních sdělení. Také s ostrými barevnými přechody mezi jednotlivými objekty v obrázku by si formát poradil vcelku obstojně (záleží na konkrétním obrázku a kvalitě segmentace).

7.2 Výkon a paměťové nároky

Vylepšit by se dala i rychlost a s ní spojené paměťové nároky. Výběr programovacího jazyka pro daný typ úlohy není nejšťastnější, zde by se lépe uplatnil nízkoúrovňový jazyk (např. *ANSI C*), nebo naprogramování alespoň nejstěžejnějších částí programu pro hardwarový akcelerátor - např. v jazyce *OpenCL* pro grafickou kartu. V dnešním moderním počítačovém světě je žádoucí kód optimalizovat pro více vláken, což vzhledem k mé snaze rozdělovat jednotlivé funkční kroky kódování a ukládání částí dat do vlastních datových struktur (např. pro každý segment vytvářím samotný objekt typu *Picture*), je proveditelné.
Také by se jednotlivé objekty daly před uložením sestupně seřadit podle velikosti a rekonstrukci obrázku provádět progresivně, to znamená nejdříve vykreslit největší objekty a poté teprve zbývající.

Kód by se dal také dále funkčně optimalizovat. Zadání bylo postupně pozměňováno, části byly řešeny modulárně a dochází ke zbytečnému průchodu seznamů atp. Například při úvodním hledání Freemanova kódu je nejprve nalezen řetězový kód pro všechny hranice segmentů v obrázku, poté jsou hranice v cyklu opět procházeny a měněny do požadovaného upraveného tvaru - tyto dva kroky lze sloučit do jednoho a v rámci hledání dalšího směru řetězový kód je možno převést do upraveného tvaru. Podobných případů lze v kódu programu najít více.
Na paměťové nároky byl při vývoji kladen důraz, lze však nalézt mnoho dalších úspor.

7.3 Vizuální kvalita a velikost

K vylepšení vlastností segmentace je možno využít kromě YC_BC_R i jiné barevné modely, např. YUV, či YP_BP_R...
Pro zlepšení vizuální kvality obrázků je možné vyzkoušet aplikaci různých druhů filtrů[11] obrazu. Např. vyhlazovací prostorové filtry, nebo filtry na zostření obrazu. Tyto filtry by dokázaly eliminovat nepříjemné šumy v obraze v podobě „tečkovaných“ ploch jednotlivých barevných segmentů.

Dále lze také vylepšit výslednou velikost obrázku a to až o 50%. Při konstrukci hranic objektů totiž sousední segmenty mají „zdvojenou“ hranici, zde se dá použít a ukládat jen hranice jednoho ze dvou sousedních segmentů.

Toto vylepšení by však vedlo k patřičnému zesložitěni rekonstrukce obrázku.

Taková vylepšení nebyla uplatněna, protože by v dané fázi produkt byl časově náročný a účelná by byla až při implementaci v reálném systému. Snažil jsem se především dokázat, že mnou navrhované řešení je funkční.

8 Závěr

V dané práci jsem se seznámil s problematikou komprese obrázků. Všechny cíle zadání jsou splněny.

Byl vytvořen nový ztrátový formát, který dokáže být na disku až 30x[1] menší než klasický formát *JPEG* se standardní 85% kvalitou (viz tab. 6.2) a výsledná komprese (respektive dekomprese) trvá maximálně v řádu jednotek vteřin (viz tab. 6.3). Zdaleka to však nejsou konečné možnosti tohoto formátu co se velikosti a kvality týče a dá se ještě ušetřit dalších až 50% velikosti (více s dalšími možnostmi rozšíření najdete v sekci 7 na straně 47). Formát způsobuje určitou degradaci výsledného obrazu, ale i u nejmenšího testovaného nastavení je stále patrná informace obsažená v obrázku. Vlastnosti řešení byly testovány prostřednictvím webové ankety a ze 42 dotázaných se 81% shoduje v názoru, že by tento formát uvítalo u zařízení, kde je kladen důraz na datové přenosy. Výborné výsledky dává řešení především u obrázků s méně plastickými objekty, které jsou vyplněny převážně jednotnou barvou. I u ostatních typů obrázků jsou výsledky velmi slušné (více v sekci 6 na straně 38).

Tato práce může sloužit k urychlení načítání webových prezentací na mobilních zařízeních, či jako další způsob komprese obrázků. Oproti *JPEG* vyniká při kompresi oblastí s prudkými změnami jasu.

[1] při nejmenší testované kvalitě nastavení

Seznam obrázků

Seznam tabulek

Literatura

[1] *Edge Detection Boundary Tracing*. EE 528 Digital Image Processing, 2005.

[2] Wikipedia, *YCbCr*. `http://en.wikipedia.org/wiki/YCbCr`, 2007 (cit. 20.2.2011).

[3] Yong Kui Liu, Borut Zalik *An effecient chain code with Huffman coding*. Pattern Recognition 38 (2005) 553-557, 2004.

[4] Richard Nock, Frank Nielsen. *Semi-supervised statistical region refinement for color image segmentation*. Pattern Recognition 38 (2005) 835-846, 2004.

[5] Jorma Rissanen, Glen G. Langdon Jr. *Arithmetic Coding. IBM Journal of Research and Develompment 23(2)*, str. 149-162, 1979.

[6] Wikipedia, *Flood fill*. `http://en.wikipedia.org/wiki/Flood_fill`, 2007 (cit. 10.12.2010).

[7] Gregory K. Wallace. *The JPEG Still Picture Compression Standard*. Digital Equipment Corporation, Maynard, Massachusetts, 1991.

[8] Hosam M. Mahmoud. *Sorting a distribution theory*. 222 Rosewood Drive, Danvers, MA 01923, ISBN 0-471-32710-7, Copyright ©2000 by John Wiley & Sons, Inc.

[9] Jiří Reif. *Metody matematické statistiky*. Fakulta aplikovaných věd, Plzeň Směrodatná odchylka (str. 23), Normální rozdělení (str. 16), ISBN 80-7043-302-7, 2004.

[10] Jiří Reif. *Metody matematické statistiky*. Fakulta aplikovaných věd, Plzeň ISBN 80-7043-302-7, 2004.

[11] Bc. Miloš Zavadil, *Obrazové filtry pro evoluční programování*. Diplomová práce, VUT Brno, 2009.

[12] Paul Bourke, *PPM / PGM / PBM Image Files*. `http://paulbourke.net/dataformats/ppm/`, 1997 (cit 10.12.2010).

[13] Stanislav Straka, *Segmentace obrazu*. Diplomová práce, 2009.

Přílohy

Příloha A

A.1 Uživatelská příručka

Popis jednotlivých kroků pro spuštění freemana.

1. Zkopírujte si z CD složku (více v Příloze C.1 na straně 64) „*aplikace*“ na root disku C, případně jinde kde uznáte za vhodné. Pro kopírování využijte například kombinaci kláves *CTRL+C* a *CTRL+V*. Nás bude nejvíce zajímat soubor s názvem „*Bakalarka.exe*“, který najdete na přiloženém CD v adresáři „*aplikace*“ a podadresářích „*bin/release*“. Nakopírujte do složky, kde se tento soubor nalézá Vaše patřičná data. Takzvaně pokud budete provádět komprimaci obrázku, tak sem nakopírujte obrázek ve formátu *PPM*[1], v opačném případě nakopírujte hlavičkovou a datovou část obrázku, jež budete dekomprimovat.

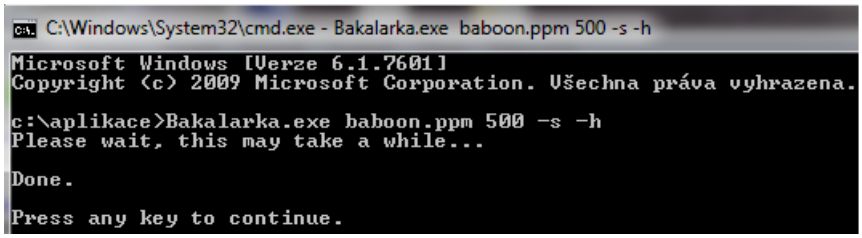
2. Dále otevřete Příkazový řádek kliknutím levým tlačítkem myši na následující tlačítka/položky: *„Nabídka Start > Programy > Příslušenství > Příkazový řádek“*.
   Do příkazového řádku napíšete příkaz: *„cd C:\aplikace“* (případně cestu k Vámi zvolené složce) a potvrdíte klávesou „Enter“.

3. Protože se přeložený program dále nachází ve složce *„bin“* s podsložkou *„Release“*, přepneme se do této složky pomocí příkazu *cd bin\Release*. Poté již stačí zadat příkaz spouštějící samotný program s patřičnými přepínači (viz tab. A.1).

4. **Komprese obrázku**
   Spouštěcí příkaz zadávejte ve tvaru: „*Bakalarka.exe obrazek.ppm [přepínače*]*“, kde „obrazek.ppm“ nahradíme obrázkem, který chceme komprimovat a přepínače nastavíme podle vlastních požadavků. Například: „*Bakalarka.exe baboon.ppm 500 -s -h*“. Uvedený příkaz způsobuje komprimaci obrázku „baboon.ppm“ při Q=500, dále provede uložení samotné segmentace do adresáře udávaném pro spuštění programu s názvem „baboon_segm.ppm“. Poté ve stejné složce naleznete výsledná komprimovaná data obrázku v podobě hlavičkové části uložené jako

[1] Nemáte-li obrázek v požadovaném formátu, uložte jej prostřednictvím jakéhokoliv zdarma dostupného prohlížeče obrázků - např. pomocí XnView

„baboon_head.jkls“ a datové části „baboon_data.jkls“. Úspěšný průběh komprese obrázku je znázorněn níže (viz obr. A.1). Také funguje příkaz „*Bakalarka.exe baboon.ppm*“, který provede komprimaci pro defaultní kvalitu nastavenou na Q=200.



Obrázek A.1: Obrázek ukazující úspěšný průběh komprimace.

5. **Dekomprese obrázku**
   Při dekomprimaci uveďte příkaz ve tvaru:
   „*Bakalarka.exe -d obrazek_head.jkls obrazek_data.jkls*“, kde pracujeme s daty vygenerovanými pomocí předchozího kroku.
   Například tedy nahradíme „obrazek_head.jkls“ a „obrazek_data.jkls“ za vygenerovaný soubor „baboon_head.jkls“ a „baboon_data.jkls“.

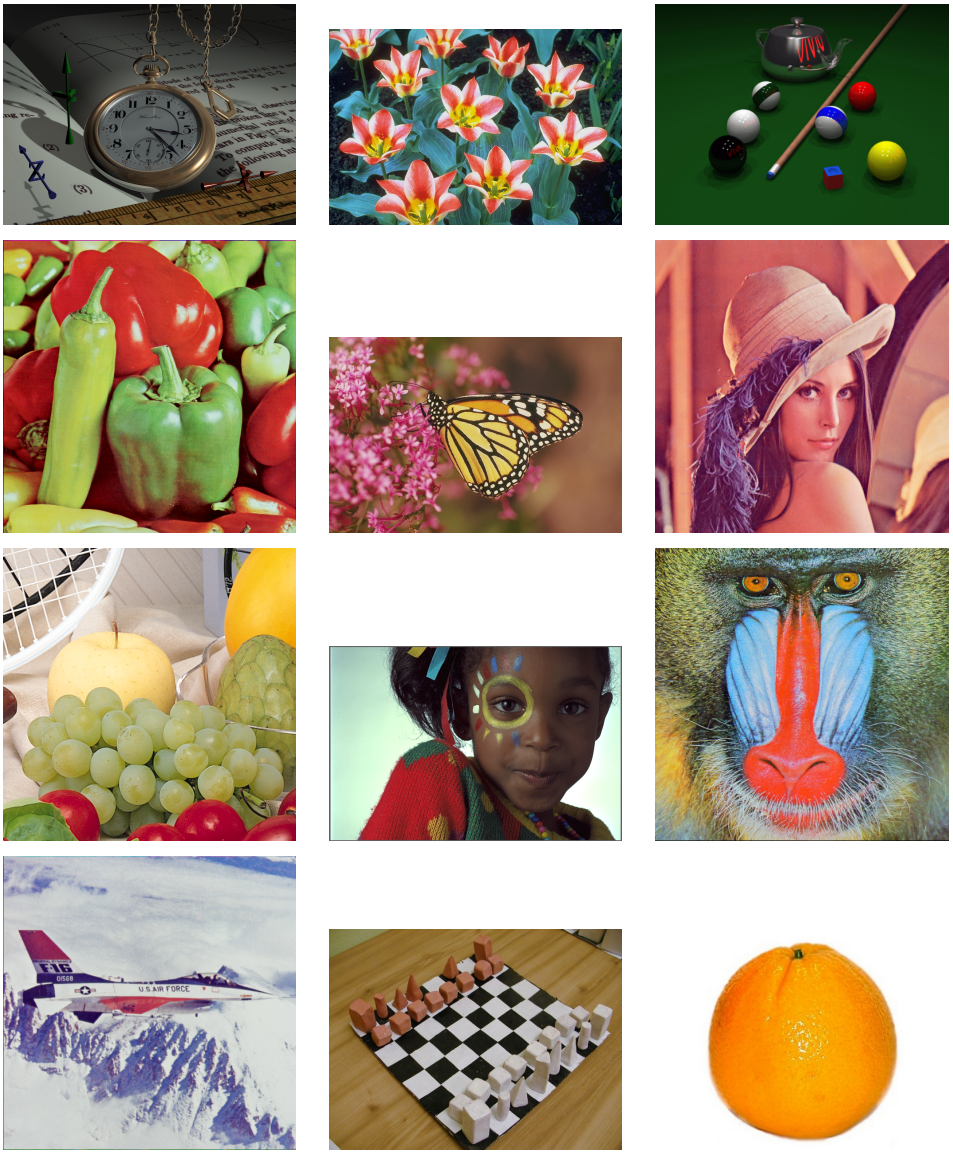
Přepínače	**Popis přepínače**
-d	nastavte, při dekompresi obrázku
Q $\in <10, 1000>$	kvalita segmentace(def.=200)
-a	aritmetické kódování (defaultně)
-h	Huffmanovo kódování
-r	použijte při komprimaci šedotónových obr.
-c	defaultně, znamená užití barevného obr.
-s	uloží segmentovaný obraz

Tabulka A.1: Seznam přepínačů.

Program je ošetřen proti chybným vstupům. Napíšete-li některý z příkazů špatně, nebo použijete-li špatný formát obrázku, bude vypsáno chybové hlášení.

Příloha B

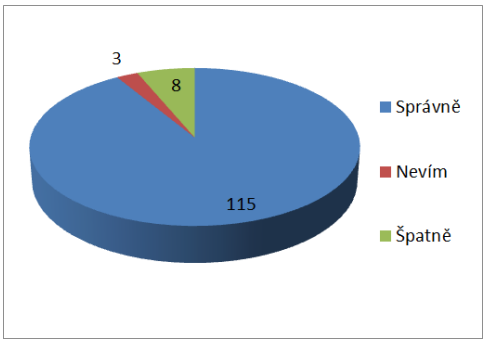
B.1 Seznam testovaných obrázků



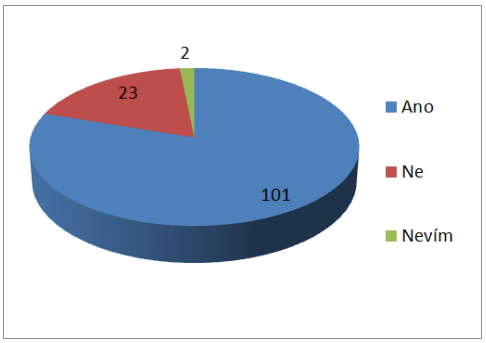
Obrázek B.1: Seznam testovaných obrázků.

B.2 Grafy výsledků ankety.

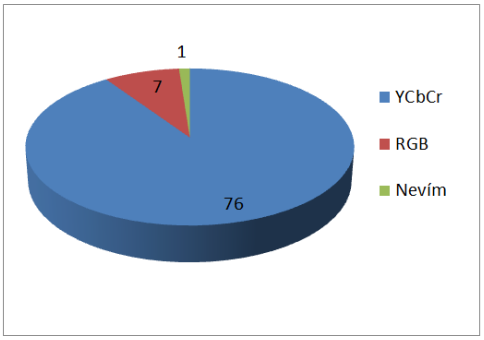
Uvádím zde některé grafy, které bebyly uvedeny v textu. Ostatní naleznete na CD (více v Příloze C.1 na straně 64).



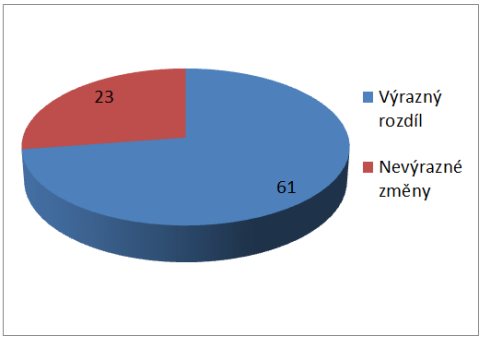
Obrázek B.2: Graf k otázce pro rozeznání kvalitnějšího obrázku. Odpověď „Správně“ znamená, že na rozpoznaný obrázek bylo Q skutečně nastaveno výše.



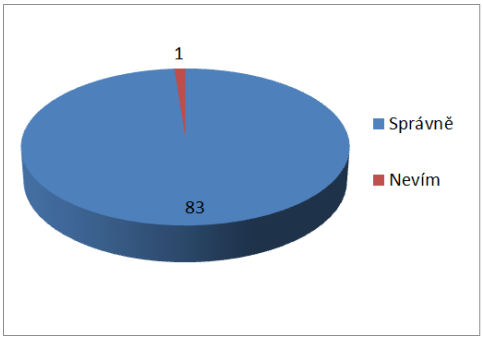
Obrázek B.3: Odpovědi na otázku: „Byla i na méně kvalitním obrázku patrná infomrace v něm obsažená?“.



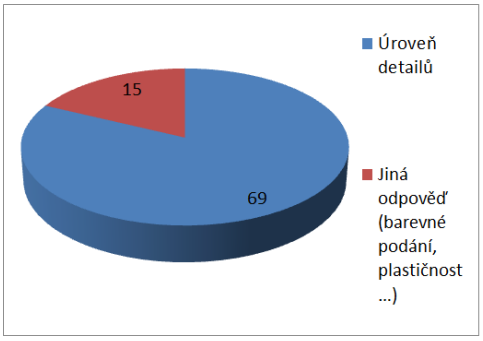
Obrázek B.4: Graf odpovědí, kde lidé měli určit kvalitnější obrázek u rozdílných barevných modelů (tzv. s využitím přechodu do YC_BC_R a bez převodu - pouze v RGB).



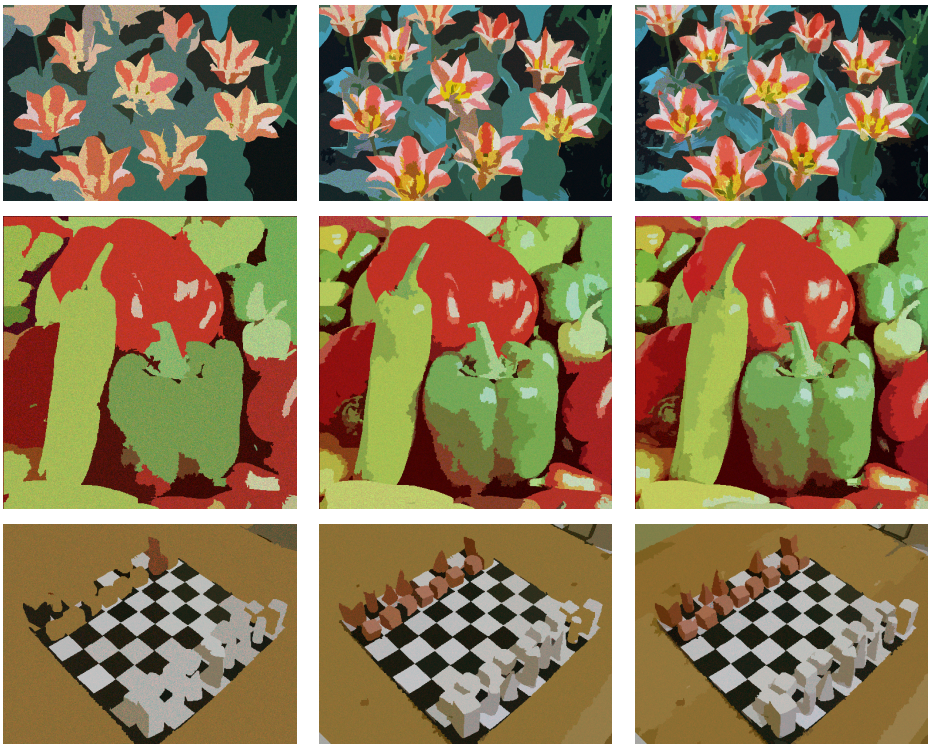
Obrázek B.5: Výsledky na otázku posouzení rozdílů kvality u 2 barevných modelů.



Obrázek B.6: Zde lidé opět určovali kvalitnější obrázek v rámci rozdílných nastavení minimálního segmentu.



Obrázek B.7: Porovnání rozdílů vyobrazených obrázků s nastavenou hladinou minimálních segmentů na 0,01% a 0,1%.



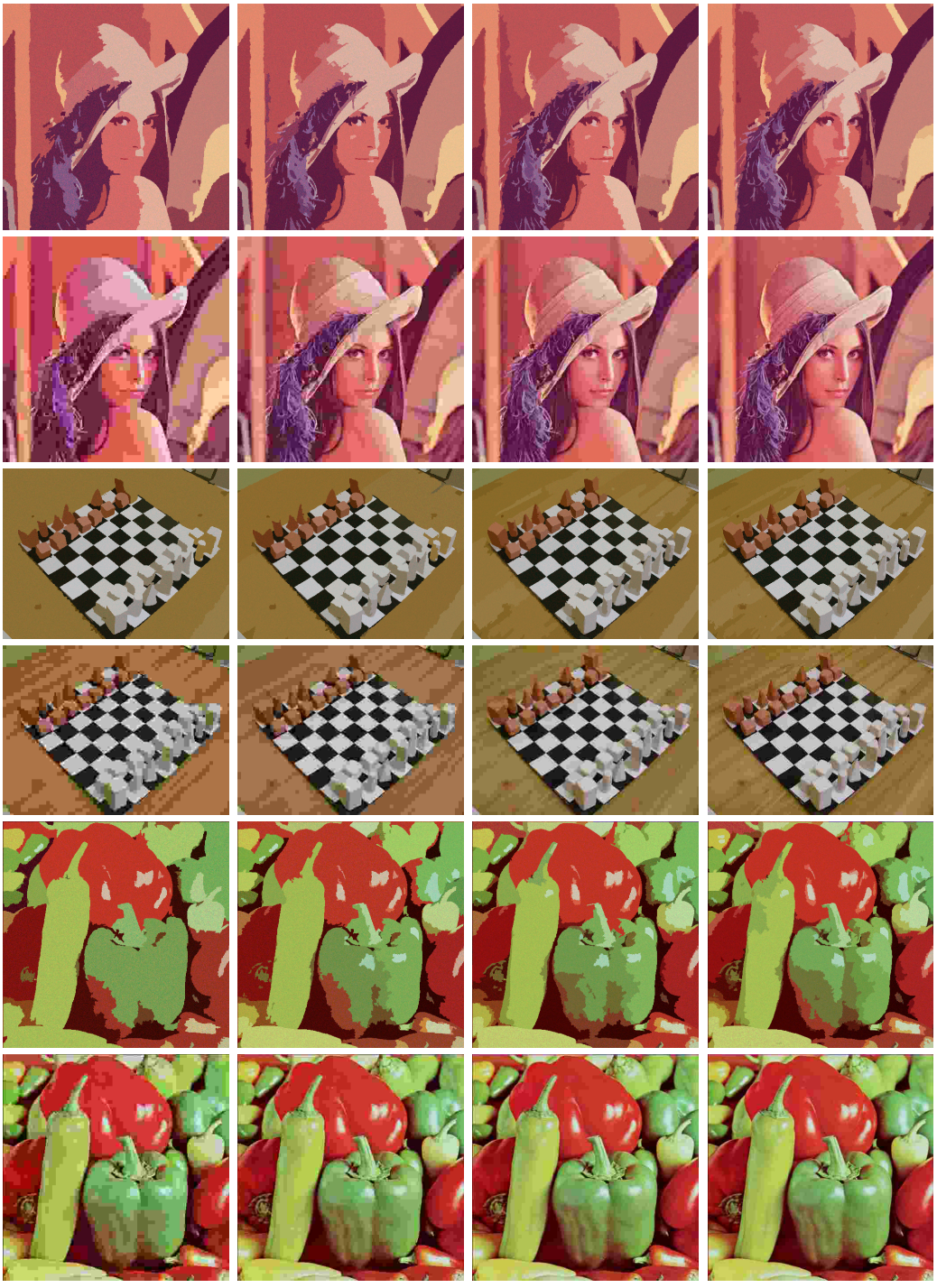
Obrázek B.8: Zleva do prava Q=32, Q=200, Q=500 s minimálním segmentem na 0,01%.

B.3 Srovnání s JPEG

Zde uvádím další srovnání s JPEG, které bohužel nebylo zahrnuto do testování.

Pro toto zhodnocení byly vybrány 3 obrázky vygenerovány tak, aby na disku zabíraly postupně 4,6,8 a 10kB - to samé platí pro obrázky v *JPEG*. Jediná vyjímka je u obrázku s šachovnicí, kde *JPEG* nebyl schopen dosáhnout patřičné 4kB hranice a zabíral celkem 5,4kB. Minimum 4kB pro obrázek jsem vybral právě kvůli omezeným možnostem *JPEG*, kdežto náš formát zvládne ještě mnohem vyšší kompresi. Více na dodávaném CD (viz C.1).

Pro porovnání také doporučuji použít dodávaný disk CD, miniatury obrázků velmi zkreslují a mohou u nich splývat detaily objektů. Otevřete-li totiž nějaký z obrázků vyobrazených níže (viz obr. B.9), bude vypadat ve skutečnosti (v originálním rozlišení) o něco hůře - to platí především u obrázku kódovaném pomocí *JPEG*, který provádí kompresi po vzorcích 8x8 pixelů. Tyto vlastnosti se u našeho formátu nevyskytují.



Obrázek B.9: Zleva doprava jsou vyobrazeny obrázky ve velikostech 4,6,8 a 10kB. Na první řádce totožného obrázku je simulován výstup našeho formátu a druhá řádka představuje obrázky ve formátu *JPEG*.

Příloha C

C.1 Obsah CD

Přiložený disk CD obsahuje složky:

- „*anketa*“ : v dané složce jsou zdrojové soubory webové ankety
- „*aplikace*“ : tato složka skrývá vlastní projekt aplikace vytvořený v programovacím prostředí *Visual Studio 2010* od Microsoftu. Je zde přeložený program i zdrojové soubory. Také se zde nachází programátorská dokumentace aplikace v angličtině.
- „*dokumentace*“ : v této složce se nachází zdrojové soubory dokumentace sázené prostřednictvím typografického rozhraní LaTeX.
- „*grafy ankety*“ : obsahuje vygenerované obrázky grafů na otázky z ankety a seznamy odpovědí.
- „*literatura*“ : některá použitá literatura v elektronické podobě.
- „*testované obrázky*“ : vygenerovaný seznam obrázků ve všech testovaných kvalitách. Také zde naleznete seznam originálních (nezměněných) obrázků.