

# Neztrátové komprimační algoritmy v počítačové grafice

Lossless Compression Algorithms in Computer Graphics

Bc. Tomáš Vogeltanz

---

Diplomová práce  
2012



Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky

---

Univerzita Tomáše Bati ve Zlíně  
Fakulta aplikované informatiky  
akademický rok: 2011/2012

## ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Tomáš VOGELTANZ**  
Osobní číslo: **A10434**  
Studijní program: **N 3902 Inženýrská informatika**  
Studijní obor: **Informační technologie**

Téma práce: **Neztrátové komprimační algoritmy v počítačové grafice**

Zásady pro vypracování:

1. Vytvořte literární rešerši na téma kompresní algoritmy v rastrové počítačové grafice. Zaměřte pozornost zejména na neztrátové algoritmy.
2. Seznamte se s nejčastěji používanými rastrovými formáty využívající neztrátový kompresní algoritmus a stručně popište jejich strukturu.
3. Navrhňte a vytvořte program, ve kterém budou implementovány neztrátové kompresní algoritmy pro rastrovou grafiku používané v současnosti.
4. Popište ovládání tohoto vytvořeného programu a jeho zdrojový kód.
5. Otestujte naprogramované kompresní algoritmy z hlediska kompresního poměru, rychlosti komprese a rychlosti dekomprese pro různé typy obrazových dat. Zjištěné výsledky porovnejte a vyhodnoťte.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

1. MURRAY, James D. a William VANRYPER. Encyklopedie grafických formátů. 2. vydání. Praha: Computer Press, 1997. ISBN 80-7226-033-2.
2. MORKES, David. Komprimační a archivační programy. Praha: Computer Press, 1998. ISBN 80-7226-089-8.
3. VEČERKA, Arnošt. Komprese dat [online]. Olomouc: Univerzita Palackého, 2008, 30.4.2008 [cit. 2011-12-18]. Dostupné z: <http://phoenix.inf.upol.cz/esf/ucebni/komprese.pdf>
4. TIŠNOVSKÝ, Pavel. Seriál Grafické formáty. Root.cz [online]. Internet Info, 7.9.2006 [cit. 2012-02-01]. Dostupné z: <http://www.root.cz/serialy/graficke-formaty/>
5. STRACHOTA, Pavel. Ukládání a komprese obrazu [online]. Praha: FJFI ČVUT, 2010, 15.9.2010 [cit. 2012-01-08]. Dostupné z: [http://saint-paul.fjfi.cvut.cz/base/public-filesystem/admin-upload/POGR/POGR1/07.ukladani\\_a\\_komprese\\_obrazu.pdf](http://saint-paul.fjfi.cvut.cz/base/public-filesystem/admin-upload/POGR/POGR1/07.ukladani_a_komprese_obrazu.pdf)

Vedoucí diplomové práce:

**Ing. Pavel Pokorný, Ph.D.**

Ústav počítačových a komunikačních systémů

Datum zadání diplomové práce:

**24. února 2012**

Termín odevzdání diplomové práce:

**21. května 2012**

Ve Zlíně dne 24. února 2012

prof. Ing. Vladimír Vašek, CSc.  
*děkan*



doc. Mgr. Roman Jašek, Ph.D.  
*ředitel ústavu*

## ABSTRAKT

Tato práce je zaměřena na neztrátové kompresní algoritmy v počítačové grafice. Nejprve jsou vysvětleny způsoby reprezentace obrazu a základní pojmy komprese. Dále jsou popsány neztrátové kompresní algoritmy a grafické formáty, které využívají neztrátovou kompresi. V rámci diplomové práce byla vytvořena aplikace, která umožňuje tyto formáty testovat. Popis této aplikace je zahrnut do úvodu praktické části. Nakonec jsou uvedeny výsledky testů neztrátových kompresních algoritmů i s jejich vyhodnocením.

Klíčová slova:

Neztrátová komprese, Testy neztrátových kompresních algoritmů, Kompresní poměr, Doba komprese, Doba dekomprese, Časová efektivita komprese, BMP, GIF, HD Photo, JPEG 2000, JPEG-LS, Lossless JPEG, OpenEXR, PCX, PNG, TGA, TIFF, WebP, JBIG, RLE, LZ77, LZW, Vlnková transformace, Prediktivní metody, PPM, Huffmanovo kódování, Shannon-Fannovo kódování, Aritmetické kódování.

## ABSTRACT

This thesis is focused on lossless compression algorithms in computer graphics. First, ways of image representation and basic compression terms are explained. Next, the lossless compression algorithms and graphic formats, that use lossless compression are described. An application, that allows to test these formats was created within diploma thesis. The description of this application is included into introduction of the practical part. Finally, the results of tests of lossless compression algorithms and their evaluating are presented.

Keywords:

Lossless Compression, Tests of Lossless Compression Algorithms, Compression Ratio, Compression Time, Decompression Time, Time Efficiency of Compression, BMP, GIF, HD Photo, JPEG 2000, JPEG-LS, Lossless JPEG, OpenEXR, PCX, PNG, TGA, TIFF, WebP, JBIG, RLE, LZ77, LZW, Wavelet Transformation, Prediction Methods, PPM, Huffman Coding, Shannon-Fano Coding, Arithmetic Coding.

Na tomto místě bych rád poděkoval panu Ing. Pavlu Pokornému, Ph.D. za jeho věcné připomínky a čas při konzultaci této práce a také všem, kteří mě při tvorbě této práce podporovali.

**Prohlašuji, že**

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

**Prohlašuji,**

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....  
podpis diplomanta

**OBSAH**

<b>ÚVOD</b> .....	<b>10</b>
<b>I TEORETICKÁ ČÁST</b> .....	<b>11</b>
<b>1 REPREZENTACE OBRAZU</b> .....	<b>12</b>
1.1 BAREVNÉ PROSTORY .....	12
1.1.1 RGB.....	12
1.1.2 RGBA.....	14
1.1.3 CMY, CMYK.....	14
1.1.4 HSV, HLS .....	15
1.1.5 YUV .....	17
1.1.6 $Y_C B_C R_C$ .....	17
1.2 REPREZENTACE RASTROVÉHO OBRAZU.....	18
1.2.1 Paleta barev .....	19
<b>2 ZÁKLADNÍ POJMY KOMPRESSE</b> .....	<b>23</b>
2.1 TYPY KOMPRESSE.....	23
2.2 HLAVNÍ PARAMETRY VÝKONU KOMPRESNÍCH ALGORITMŮ.....	25
<b>3 NEZTRÁTOVÉ KOMPRESNÍ ALGORITMY</b> .....	<b>27</b>
3.1 PÍXELOVÉ ZHUŠŤOVÁNÍ.....	27
3.2 RLE .....	28
3.2.1 Bitová úroveň kódování .....	31
3.2.2 Bytová úroveň kódování .....	31
3.2.3 Pixelová úroveň kódování.....	32
3.2.4 Tříbytové kódování .....	33
3.2.5 Vertikální replikační pakety .....	34
3.3 LZ77 .....	35
3.4 LZW .....	37
3.5 HUFFMANOVO KÓDOVÁNÍ .....	40
3.5.1 CCITT kódování .....	43
3.6 SHANNON-FANOVO KÓDOVÁNÍ .....	45
3.7 ARITMETICKÉ KÓDOVÁNÍ .....	47
3.8 JBIG.....	50
3.9 JBIG 2.....	52
3.10 VLNKOVÁ TRANSFORMACE .....	54
3.10.1 Diskrétní vlnková transformace .....	55
3.10.2 Celočíslná vlnková transformace .....	57
3.11 PREDIKTIVNÍ METODY .....	58
3.11.1 PPM.....	60

3.12	BURROWS-WHEELEROVA TRANSFORMACE .....	62
<b>4</b>	<b>FORMÁTY VYUŽÍVAJÍCÍ NEZTRÁTOVOU KOMPRESI.....</b>	<b>65</b>
4.1	BMP .....	65
4.2	GIF .....	68
4.3	PCX.....	75
4.4	PNG.....	79
4.5	TGA.....	87
4.6	TIFF.....	93
4.7	JPEG 2000.....	99
4.8	HD PHOTO .....	103
4.9	JPEG-LS.....	104
4.9.1	Lossless JPEG .....	107
4.10	OPENEXR .....	108
4.11	WEBPLL.....	111
<b>II</b>	<b>PRAKTICKÁ ČÁST.....</b>	<b>113</b>
<b>5</b>	<b>APLIKACE PRO TESTOVÁNÍ KOMPRESNÍCH ALGORITMŮ.....</b>	<b>114</b>
5.1	MOŽNOSTI APLIKACE.....	114
5.1.1	Menu Soubor.....	115
5.1.2	Menu Obrázek.....	120
5.1.3	Menu Nápověda .....	122
5.2	POPIS ZDROJOVÝCH KÓDŮ APLIKACE.....	123
5.2.1	Třída Obrazek.....	124
5.2.2	Třída FreeImageRozhraniClass.....	126
5.2.3	Třída MainFrame.....	127
<b>6</b>	<b>TESTOVÁNÍ KOMPRESNÍCH ALGORITMŮ .....</b>	<b>130</b>
6.1	FOTOGRAFIE.....	131
6.1.1	Fotografie formátu JPEG .....	131
6.1.2	Fotografie formátu RAW .....	133
6.1.3	Fotografie v odstínech šedi .....	135
6.1.4	Fotografie s vysokou dynamikou barev.....	136
6.1.5	Fotografie s vysokou dynamikou barev v odstínech šedi.....	138
6.2	OBRÁZKY .....	140
6.2.1	Obrázky ve 24 bitové hloubce.....	140
6.2.2	Obrázky v 8 bitové hloubce.....	142
6.2.3	Obrázky ve 4 bitové hloubce.....	144
6.2.4	Obrázky v 1 bitové hloubce.....	146
6.2.5	Obrázky v odstínech šedi .....	148
6.3	TEXT .....	150
6.3.1	Text v 1 bitové hloubce.....	150
6.3.2	Text v odstínech šedi.....	153

---

<b>ZÁVĚR .....</b>	<b>155</b>
<b>ZÁVĚR V ANGLIČTINĚ.....</b>	<b>157</b>
<b>SEZNAM POUŽITÉ LITERATURY.....</b>	<b>159</b>
<b>SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK .....</b>	<b>167</b>
<b>SEZNAM OBRÁZKŮ .....</b>	<b>172</b>
<b>SEZNAM TABULEK.....</b>	<b>174</b>
<b>SEZNAM PŘÍLOH.....</b>	<b>176</b>

## ÚVOD

Tato práce je zaměřena na neztrátové kompresní algoritmy v počítačové grafice - konkrétně na rastrové obrázky ve 2D. Na úvod jsou popsány možnosti reprezentace obrazu včetně typů barevných prostorů a palety barev.

V rámci druhé kapitole jsou vysvětleny pojmy používané v souvislosti s kompresí. Zde jsou uvedeny základní typy kompresí a parametry, které se vyskytují při vyhodnocení kvality kompresních algoritmů.

Ve třetí kapitole jsou charakterizovány neztrátové kompresní algoritmy. Zastoupeny zde jsou jak starší komprese jako Huffmanovo kódování, RLE, LZ77, LZW, tak komprese novější, ke kterým patří JBIG, vlnková transformace a prediktivní metody.

Čtvrtá kapitola objasňuje problematiku formátů, které využívají neztrátové kompresní algoritmy. Jsou zde popsány struktury těchto formátů a zmíněny neztrátové kompresní algoritmy, které tyto formáty využívají. Tato kapitola obsahuje formáty, jejichž použití je v současnosti již méně časté (např. PCX, TGA), ale také ty, které se hojně využívají (např. GIF, PNG), a ty novější, které si stále hledají cestu do podvědomí běžného uživatele (např. JPEG 2000, JPEG-LS, HD Photo).

Dále je v práci popsána vytvořená aplikace pro testování neztrátových kompresních algoritmů. V páté kapitole je znázorněn výčet možností této aplikace a je upřesněna funkcionality nejdůležitějších tříd.

Na konci práce jsou uvedeny výsledky testů neztrátových kompresních algoritmů pro dané typy obrázků. Tyto výsledky jsou vyhodnoceny a na jejich základě jsou kompresní algoritmy vzájemně porovnány.

Na CD jsou umístěny dvě verze této práce - standardní (určená pro tisk) a rozšířená. Rozšířená verze detailněji popisuje reprezentaci obrazu, neztrátové kompresní algoritmy, formáty obrazových souborů a obrázky použité při testech algoritmů. Dále lze na CD najít zdrojové kódy aplikace pro testování kompresních algoritmů spolu se spustitelným souborem a použitými externími komponenty.

## **I. TEORETICKÁ ČÁST**

## 1 REPREZENTACE OBRAZU

Reprezentace obrazu se v počítačové grafice dělí na rastrovou a vektorovou [1].

Rastrová reprezentace obrazu popisuje obraz různými způsoby, především pomocí matice pixelů. U matice pixelů vzniká problém konečného rozlišení (např. obrázek nelze přiblížit bez ztráty kvality) a vysoké paměťové náročnosti (resp. větší velikosti souboru). Kvůli velké velikosti souboru s rastrovým obrázkem je také snaha o kompresi rastrových dat. [2] Rastrovou grafikou lze však zobrazit i složité předlohy a práce s těmito daty je rychlejší [3].

Další možností reprezentace rastrového obrazu je pomocí kvadrantového stromu. Zde se využívá koherence ve vodorovném a svislém směru. Pomocí této metody se úsporně kódují větší souvislé plochy jedné barvy (resp. menšího počtu barev). Princip této metody je adaptivní - přispůsobuje se datům. Pokud je barva všech pixelů obrazu stejná uloží se informace o této barvě, pokud ne celý obraz je rozdělen na kvadranty. Následně když je barva pixelů v kvadrantu stejná, tak dojde k zápisu této barvy, když ne dojde opět k rozdělení tohoto kvadrantu na menší kvadranty a proces se rekurzivně opakuje. [4]

Vektorová reprezentace obrazu je popis geometrických objektů a jejich vlastností (polohy, barvy, překrývání atd.). Díky tomu, že obsahuje pouze parametry objektů (co, kam a jak se má vykreslit), nezávisí na rozlišení zobrazovacího zařízení a obrázek lze libovolně škálovat při zachování kvality. [2] U vektorového formátu se komprese příliš nepoužívá, neboť zde má malý efekt. Pokud je na soubor ve vektorovém formátu komprese použita, tak dochází ke kompresi celého souboru a komprese musí být neztrátová. [3]

### 1.1 Barevné prostory

Barevné prostory určují, jakým způsobem je definována cílová barva [3].

#### 1.1.1 RGB

Tento barevný prostor se v grafických formátech používá nejčastěji [3].

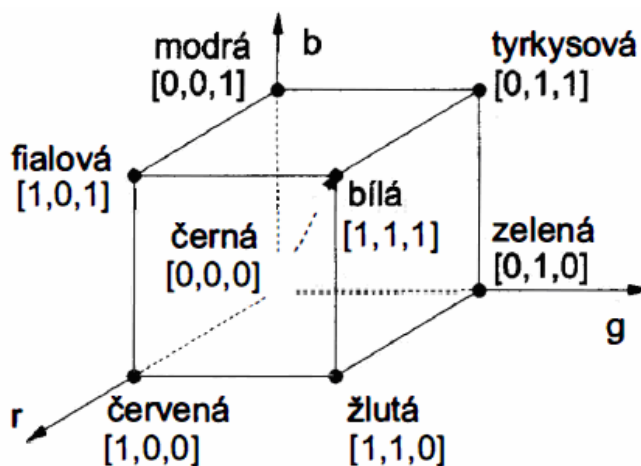
Základní vlastností tohoto barevného prostoru je součtové, aditivní skládání barev - čím více barev je zkombinováno (sečteno), tím světlejší je výsledek [1]. Aditivní barevné prostředí nepotřebuje žádné vnější světlo [3]. Barvy lze vyjádřit trojicí (barevným vektorem), jejíž složky nabývají hodnot z intervalu  $\langle 0 ; 1 \rangle$ . Bývají uváděny i v

celočíselném rozsahu 0-255, což odpovídá kódování každé ze složek RGB (Red - červená, Green - zelená, Blue - modrá) v jednom Bytu. Hodnota 0 znamená, že složka není zastoupena, maximální hodnota 255 indikuje, že složka nabývá své největší intenzity. [1]

Počet barevných odstínů, který lze reprezentovat trojicí Bytů je  $256^3$  (resp.  $2^{24}$ ) = 16777216. Kombinací červené, zelené a modré barvy zobrazenou displejem v plné intenzitě, lze získat barvu bílou. [1] Naopak při nulové intenzitě všech těchto barev je možné obdržet černou barvu [3]. Šedá barva se vytvoří kombinací všech tří barev se shodnou intenzitou. Pokud je potřeba převést různobarevný obraz na šedý, nelze jeho barvy nahradit odstíny šedi získanými průměrem ze tří základních barev. Lidské oko vnímá různým způsobem intenzitu jednotlivých barevných složek, a proto se používá pro výpočet jasů empirický vztah: [1]

$$I = 0,299 R + 0,587 G + 0,114 B$$

Konkrétní reprezentace barvy může být různorodá. Od jednobitové informace rozlišující pouze mezi stavy černá a bílá, přes osmibitové číslo označující stupeň šedi až po různé zápisy barevných složek. Reprezentace barevných složek RGB ve dvou Bytech (5-6-5 bitů) se označuje high color, zápis ve třech Bytech (8-8-8 bitů) true color. Někdy jsou barvy vyjádřeny nepřímo - pomocí čísla, odkazujícího do tabulky, zvané paleta. [1]



Obr.1 Geometrická reprezentace prostoru RGB [1]

### 1.1.2 RGBA

Zkratka RGBA (resp. RGBa) je používána pro vyjádření skutečnosti, že barevný obraz zapsaný v prostoru RGB je doplněn informací o průhlednosti. Každý barevný bod takového obrazu s sebou nese skalární údaj (např. v rozmezí 0-1), který určuje, v jakém rozsahu pokrývá barva plochu obrazového bodu. Hodnota 0,0 znamená neprůhledný barevný bod, maximální hodnota 1,0 zcela průhledný. Při zobrazení samotného obrazu nemá složka  $\alpha$  ( $\alpha$ -kanál) význam. Pojem RGBA neznámá změnu barevného prostoru, nýbrž přidání další informace. Složka  $\alpha$  se ukládá do rozsahu jednoho i více Bytů. Používá se zejména při kombinování více obrazů do jednoho celku. [1]

### 1.1.3 CMY, CMYK

Využívá subtraktivní skládání barev, neboli smísení jednotlivých barevných pigmentů, přičemž každé přidání pigmentu vytvoří tmavší barvu. Složením všech barev vznikne černá, což je přesně opačná situace oproti aditivnímu skládání barevného světla. [1] Subtraktivní prostředí je prostředí, které odráží světlo, a proto pro jeho vnímání je potřeba vnějšího zdroje světla [3].

CMY obsahuje tři základní barvy: tyrkysovou neboli modrozelenou (Cyan), fialovou (Magenta) a žlutou (Yellow). [1]

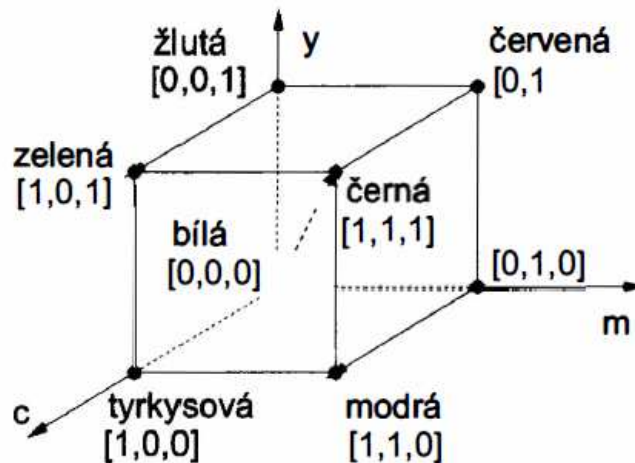
Pokud je barevný vektor vyjádřen v prostoru RGB tříprvkovou maticí  $\begin{bmatrix} r & g & b \end{bmatrix}$ , určíme vektor  $\begin{bmatrix} c & m & y \end{bmatrix}$  v prostoru CMY odčítáním matic: [1]

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Základní barevné pigmenty C, M a Y nesmí být dokonale krycí, neboť nové barvy vznikají vzájemným překrýváním [1]. Protože barvy nejsou v praxi ideální přítomnost všech tří barev při tisku nevznikne požadovaná černá, ale tmavě hnědá [3]. Navíc není ekonomické skládat černou barvu ze tří jiných barev. Z těchto důvodů se černá tiskne samostatně. Lze ji použít i ke ztmavení ostatních barev. Od prostoru CMY se tak v polygrafii přechází k prostoru CMYK přidáním černé (black) jako čtvrté základní barvy. Velikost černé složky pro daný barevný bod lze získat jako minimální hodnotu ze složek C, M a Y, které

poté snížíme o  $K$ . Tento jednoduchý postup získání hodnoty  $K$  však bývá v praxi většinou upravován s ohledem na použitá barviva a způsoby jejich mísení. [1]

V mnoha reálných systémech jsou barevné komponenty CMYK určeny jako procenta v rozsahu 0-100. [3]



Obr.2 Geometrická reprezentace prostoru CMY [1]

#### 1.1.4 HSV, HLS

Oba prostory HSV i HLS definují barvu trojicí složek, které nepředstavují základní barvy. Třemi hlavními parametry prostoru HSV jsou barevný tón (Hue), sytost (Saturation) a hodnota jasu (Value). Barevný tón označuje převládající spektrální barvu, sytost určuje příměs jiných barev a jas je dán množstvím bílého (bezbarvého) světla. Pro zobrazení prostoru se nepoužívá krychle, nýbrž šestiboký jehlan (Obr.3 vlevo), jehož vrchol leží v počátku soustavy souřadnic HSV. Souřadnice  $S$  a  $V$  se mění od 0 do 1, souřadnice  $H$  reprezentuje úhel a nabývá hodnot z intervalu  $\langle 0^\circ ; 360^\circ \rangle$ . Vrchol jehlanu představuje černou barvu. Jas roste směrem k podstavě, střed podstavy reprezentuje bílou barvu. Sytost odpovídá relativní vzdálenosti bodu od osy jehlanu. Dominantní barvy (se sytostí jedna) tedy leží na plášti, čisté barvy jsou na obvodu podstavy. Při pohybu po obvodu ve stejné výšce od základny se postupně mění barevný tón - sytost a jas zůstávají nezměněny. [1]

Prostor HSV vykazuje některé nedostatky, které sice nejsou zásadního charakteru, nicméně mohou ztěžovat práci s přesným určením barvy. Jedním z nedostatků je jehlanovitý tvar,

který způsobuje, že ve vodorovném řezu se musí bod o konstantní hodnotě  $S$  pohybovat při změně  $H$  po dráze ve tvaru šestiúhelníku, nikoliv po kružnici, jak by bylo přirozené. Dalším záporným jevem je nesymetrie prostoru z hlediska jasů. Tyto nedostatky odstraňuje prostor HLS, jehož geometrie je uvedena na obrázku *Obr.3* vpravo. [1]

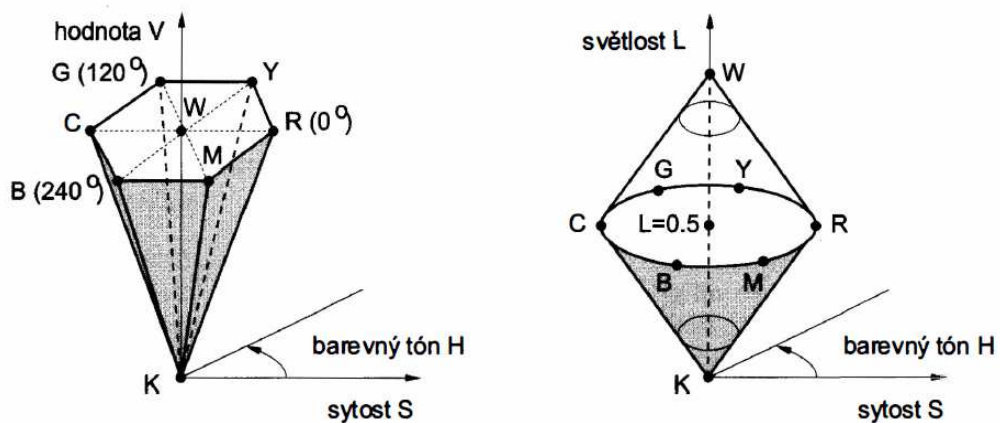
Název prostoru HLS je odvozen z pojmů barevný tón (Hue), světlost (Lightness) a sytost (Saturation). Prostor HLS je obdobou prostoru HSV, v němž byl jehlan nahrazen dvojicí kuželů. [1]

Barevný tón je opět vyjádřen úhlovou hodnotou, světlost se mění od 0 (černá v dolním vrcholu) do 1 (bílá v horním vrcholu). Sytost nabývá na povrchu kuželů hodnoty 1 a klesá na 0 směrem k ose kuželů. Nejjasnější čisté barvy mají tedy souřadnice  $S = 1$  a  $L = 0,5$  a leží na obvodu podstav kuželů. [1]

Tvar prostoru HLS plně odpovídá skutečnosti, že nejvíce různých barev vnímáme při průměrné světlosti (oblast podstav). Schopnost rozlišit barvy klesá jak při velkém ztmavení, tak při přesvětlení (oblasti obou vrcholů kuželů). Další dobrá vlastnost prostoru HLS spočívá v analogii míchání barev přidáváním černých a bílých pigmentů k základním spektrálním barvám. [1]

Prostory HSV a HLS umožňují postupně měnit barevné charakteristiky při zachování ostatních typických vlastností barvy. [1]

Převod barvy z prostorů HLS a HSV do prostoru RGB má charakter algoritmu. Převod není prostým zobrazením. Pro barvy ležící na ose jehlanu HSV či kuželu HLS není hodnota  $H$  jednoznačně určena. V takových případech se zavádí zvláštní hodnota nedefinováno. Například při velikosti  $S = 0$  nesmí nabývat složka  $H$  žádné platné hodnoty a může být pouze nedefinována. [1]



Obr.3 Geometrická reprezentace prostoru HSV (vlevo) a HLS (vpravo) [1]

### 1.1.5 YUV

Základní prostor YUV se používá pro přenos televizních signálů v normě PAL. Do stejné rodiny patří další prostory - YIQ pro americkou televizní normu NTSC a prostor  $Y C_B C_R$  pro normu SECAM. Jejich společným rysem je oddělení jasové (luminanční) složky od barevných informací (chrominance). Toto uspořádání dovoluje přímo využít stejný jasový signál Y jak pro barevné, tak i černobílé obrazy. Zbývající dva signály nesou informace o velikosti barevných složek obrazu. [1]

Prostor YUV je někdy označován jako  $[V, B-Y, R-Y]$ , resp. číselně  $[Y, 0,493(B-Y), 0,877(R-Y)]$ . Z prostoru RGB je možné získat hodnoty YUV maticovým násobením: [1]

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0,299 & 0,587 & 0,114 \\ -0,141 & -0,289 & 0,437 \\ 0,615 & -0,515 & -0,100 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

### 1.1.6 $Y C_B C_R$

S tímto prostorem se v počítačové grafice lze setkat při zápisu rastrových obrázků ve formátu JPEG. [1]

Podle standardu CCIR-601 má hodnota Y být v intervalu  $\langle 0,0 ; 1,0 \rangle$  a hodnoty  $C_B, C_R$  v intervalu  $\langle -0,5 ; 0,5 \rangle$ . Pokud mají být při počítačovém zpracování uloženy složky  $C_B$  a  $C_R$  jako celá čísla v rozsahu 0-255, musí se nejprve posunout do kladné poloosy přičtením konstanty. [1]

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0,299 & 0,587 & 0,114 \\ -0,1687 & -0,3313 & 0,5 \\ 0,5 & -0,4187 & -0,0813 \end{bmatrix} \cdot \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Zkráceně lze definovat hodnoty  $C_B$  a  $C_R$  jako: [1]

$$C_B = 0,5643 \cdot (B - Y)$$

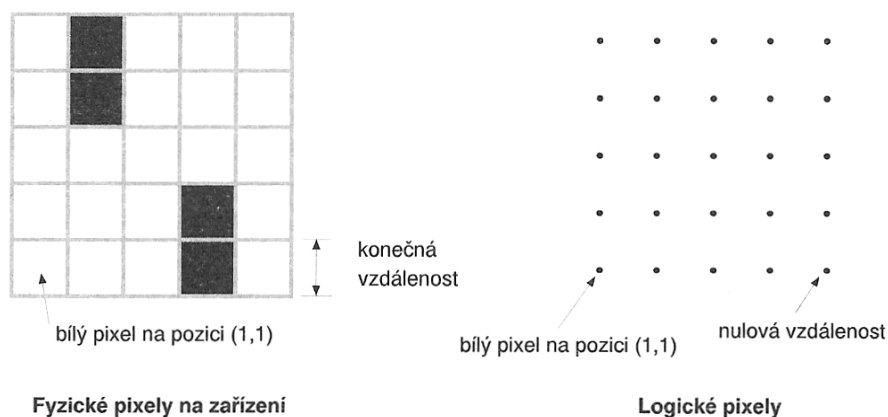
$$C_R = 0,7133 \cdot (R - Y)$$

## 1.2 Reprezentace Rastrového obrazu

V počítačové grafice jsou pozice dat uloženy jako matematické souřadnice. Rozlišují se zde pixely fyzické a logické. [3]

Fyzické pixely jsou body, které jsou používány pro zobrazování na výstupním zařízení. Tyto pixely jsou přímo ovládány hardwarem výstupního zařízení. [3]

Narozdíl od fyzických pixelů jsou logické pixely matematické body. Specifikují polohu, ale fyzicky nezabírají žádnou plochu. Mapování mezi logickými pixely v rastrových datech musí odpovídat skutečné velikosti a uspořádání fyzických pixelů. [3]



Obr.4 Fyzické a logické pixely [3]

Počet bitů, které reprezentují daný pixel rovněž určuje počet barev, kterých může pixel nabývat. Čím větší je počet bitů, tím větší je i počet barev. Více bitů na jeden pixel také znamená více prostoru v paměti a na disku, který zaberou rastrová obrazová data. [3]

Vyjádření barevných složek pomocí 3 Bytů je v současnosti nejběžnější. Používají se ale i jiná kódování, např. 12 nebo 16 bitů na barevný kanál. [1]

Pokud je každý pixel na obrazovce popsán jediným bitem, říká se, že je takový obraz monochromatický neboli černobílý. Toto označení je poněkud zavádějící, protože binární informace nemusí reprezentovat právě černou a bílou barvu, ale dvě libovolné barvy. [1]

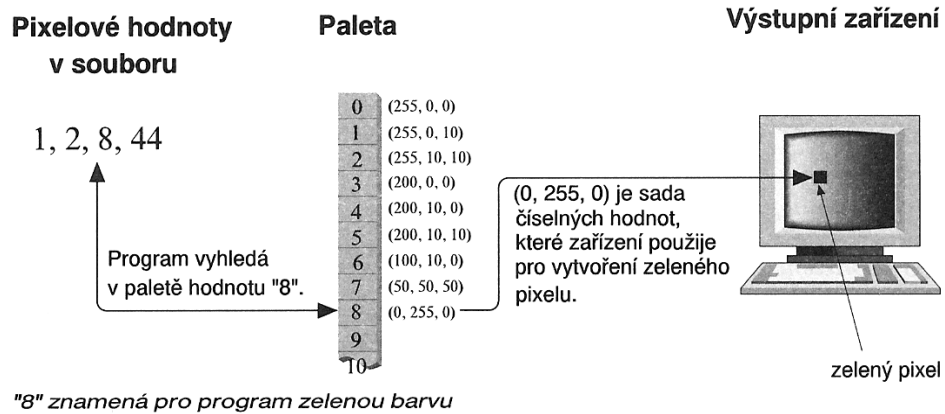
Indexový mód je spojen s používáním tzv. barevné palety, neboli mapy barev. U takového obrazu nereprezentuje hodnota pixelu přímo barvu, ale je ukazatelem do tabulky - barevné palety. [1] Indexovým módem lze ušetřit místo na disku, protože tři hodnoty barev o celkové velikosti 3 Byty jsou nahrazeny jedním indexem o velikosti 1 Byte - data jsou tak zmenšena trojnásobně. [3]

Další možností je reprezentace obrazu v odstínech šedi. Každý bod v obraze reprezentuje buď přímo odstín šedi (static grey), nebo je opět odkazem do palety (greyscale). [1]

### 1.2.1 Paleta barev

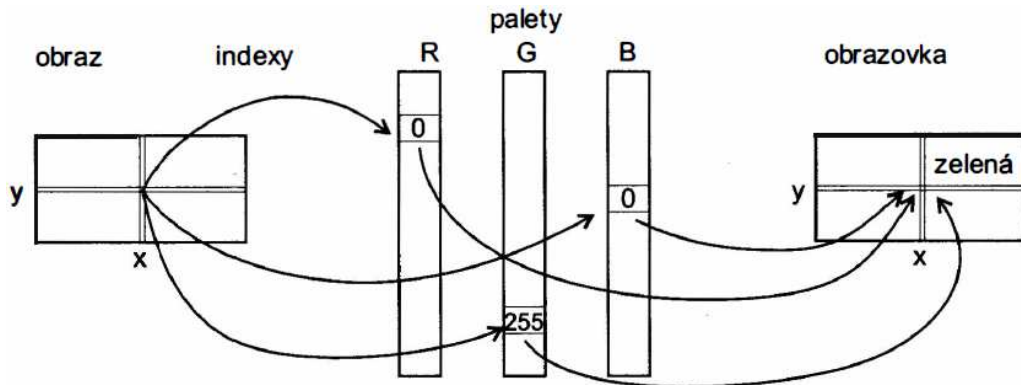
Barevná paleta je převodní tabulka, která určuje konkrétní barvu pixelu daným indexem [1]. Například osmibitová hodnota pixelu může reprezentovat 256 ( $2^8$ ) různých barev a je tedy doprovázena 256 prvkovou paletou. Pokud má předloha méně barev než je jejich maximální počet v paletě, potom jsou všechny nepoužité prvky v ní nastaveny na nulu. Nevyužité prvky v paletě většinou nejsou uspořádány ve slucích podle nějakého systému, také většinou nezačínají na nulové pozici. [3]

Paleta označovaná jako 8-8-8 reprezentuje barvu ve třech Bytech, po jednom pro každý barevný kanál. Tím je určeno, že každý bod obrazu může nabývat některé z 256 barev, které se vybírají z celkového množství  $2^{24}$  barev. Tento způsob přiřazení barev bývá také označován jako pseudo color. Barevná paleta je buď součástí obrazu nebo je vytvářena až při jeho zobrazování. [1] Princip použití indexového módu lze nalézt na *Obr.5*. [3]



Obr.5 Princip použití indexového módu pro určení barev [3]

Barevný mód označovaný jako direct color pracuje s RGB hodnotami, které neslouží přímo k zobrazení, nýbrž jsou odkazem do barevných palet pro každou jednotlivou barevnou složku. Při zobrazení musí tedy být k dispozici tři palety, pro každý barevný kanál jedna. Tento způsob je výhodný, protože umožňuje snadnou změnu všech barev, bez změny hodnoty pixelů v obrazu a využívá se například při gama korekci. Princip použití direct módu je možné zhlédnout na Obr.6. [1]



Obr.6 Princip použití direct módu pro určení barev [1]

U pixelově orientované palety je způsob ukládání hodnot do souboru většinou v RGB nebo BGR uspořádání: [3]

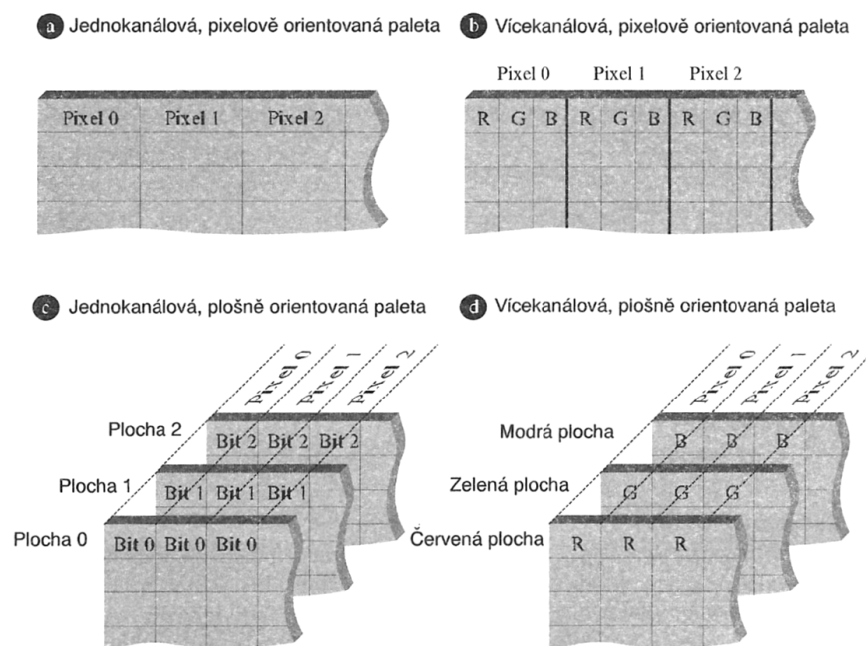
(RGBRGBRGBRGB...) nebo (BGRBGRBGRBGR...)

V plošně orientované paletě jsou barevné komponenty pixelu odděleny: [3]

(RRRRR...GGGGG...BBBBB...) nebo (BBBBB...GGGGG...RRRRR...)

Palety lze rozdělit na: (viz také *Obr.7*) [3]

- Jednosložková pixelově orientovaná paleta - obsahuje jednu hodnotu pixelu na každý prvek.
- Vícesložková pixelově orientovaná paleta - obsahuje jednu hodnotu pixelu na každý prvek a každý pixel obsahuje dvě nebo více barevných složek dat.
- Jednosložková plošně orientovaná paleta - obsahuje jeden pixel na index a jeden bit na plochu.
- Vícesložková plošně orientovaná paleta - obsahuje jednu hodnotu barevné složky na každý prvek.



*Obr.7 Typy palet* [3]

Paleta může být: [2]

- Pevně daná (např. standardní VGA paleta)
- Adaptivní (vhodně přizpůsobená obrázku)

Nejčastěji se používá u obrázků s počtem bitů na pixel menší než 8 bitů, a to vzhledem k velikosti palety, která by mohla přesáhnout velikost samotných obrazových dat (např.  $2^{24} = 16\,777\,216 * 3 = 50\,331\,648 \text{ B} = 48 \text{ MB}$ ) [3].

U pevné palety může nastat problém, kdy chceme pixelu o obecné barvě přiřadit barvu z palety. Pokud obsahuje paleta  $n$  barev o složkách  $[R_i ; G_i ; B_i]$  a vzdálenost barev měříme euklidovskou mírou, tak barvu  $[R ; G ; B]$  nahradíme barvou palety s indexem  $i$ : [2]

$$\arg \min_{i \in N_n} \sqrt{(R - R_i)^2 + (G - G_i)^2 + (B - B_i)^2}$$

Oblasti prostoru RGB nahrazené  $i$ -tou barvou palety jsou tzv. Voroného diagramy [2].

## 2 ZÁKLADNÍ POJMY KOMPRESSE

Kompresse (resp. komprimace) je proces, který se využívá pro zredukování fyzické velikosti bloku informací [3]. Jinými slovy: Soubory jsou zakódovány do takové podoby, kdy je jejich velikost menší než velikost před kompresí [5]. Kódování znamená způsob reprezentace dat při jejich uložení v souboru, paměti apod. [6]

Každý kompresní algoritmus je navržen tak, že hledá a využívá pro kompresi dat určitý řád v uložených datech. Tímto řádem může být opakování sekvencí znaků, frekvence výskytu jednotlivých znaků, identifikace dlouhých bloků stejných dat a další. [5]

Data různého charakteru vyžadují rozdílný přístup k jejich kompresi. Je zřejmé, že grafická data se svým charakterem budou lišit např. od textových, a že rozdílnost těchto dat způsobí rozdílné možnosti aplikovatelnosti některých kompresních algoritmů. Např. RLE lze více využít u grafických formátů než u textových - v textu se málokdy objevují stejné znaky za sebou. [5]

### 2.1 Typy komprese

Kompresní algoritmy se v základu dělí na neztrátové a ztrátové.

Neztrátová komprese je způsob komprese, při které nedochází ke ztrátě informace - při dekompresi dostáváme stejná data, jako před kompresí. Místo termínu bezztrátová komprese se také používají označení přesná komprese nebo vratná komprese. [6] Všeobecně tyto metody nedosahují tak dobrých kompresních poměrů jako metody ztrátové, ale používají se i jako pomocné algoritmy ke kódování obrazu a zvuku [7].

Ztrátová komprese je způsob komprese, při které jsou výchozí hodnoty poněkud pozměněny nebo některé méně významné hodnoty jsou zanedbány, aby se dosáhlo vyššího kompresního poměru. Dekompresí dostáváme v tomto případě jiné hodnoty, než před kompresí. Ztrátové metody mají uplatnění v kompresi obrazu a zvuku. [6]

Další možnosti dělení kompresních algoritmů podle [3]:

- Fyzická × Logická
- Symetrická × Asymetrická
- Neadaptivní × Adaptivní × Semiadaptivní

Logická komprese používá logické substituce sekvence znaků jinou, úspornější řadou. Konkrétním příkladem jsou zkratkovaná slova jako Čedok (nahrazující někdejší plný název Československá dopravní kancelář) nebo Svazarm (Svaz pro spolupráci s armádou). [5]

Fyzická komprese probíhá bez ohledu na logiku dat, se kterými se manipuluje. Vytváří se nová sekvence znaků (Bytů, bitů apod.), jejíž vztah k původním datům lze rozpoznat výhradně s použitím dekompresního algoritmu. Bez znalosti tohoto dekompresního algoritmu je informační hodnota komprimovaných dat nulová. [5] Algoritmy v počítačové grafice jsou založeny výhradně na fyzické kompresi [3].

O symetrickou kompresi se jedná, pokud je doba (a tím většinou i počet a druh operací) potřebná pro kompresi i dekompresi dat přibližně stejná [5].

U asymetrické komprese není čas potřebný pro kompresi a dekompresi stejný. Většina kompresních algoritmů provede větší množství operací při kompresi dat. Asymetrické algoritmy, jejichž práce je delší při dekompresi, nejsou tolik rozšířené. [5]

Neadaptivní algoritmy jsou určeny výhradně pro kompresi specifického druhu dat. Většinou obsahují předdefinované slovníky nebo řetězce znaků, o kterých je známo, že jejich pravděpodobnost výskytu v souborech dat je vysoká. Použití neadaptivního algoritmu na vhodný druh dat je velice účinné co do dosaženého kompresního poměru, ale i času potřebného pro kompresi a dekompresi dat. Konkrétním příkladem neadaptivního kompresního algoritmu je tzv. Huffmanovo (resp. CCITT) kódování. [5]

Adaptivní algoritmus je naproti tomu schopen dosáhnout určité nezávislosti na komprimovaných datech. Takové algoritmy neobsahují žádné statické slovníky řetězců. Algoritmy si budují tyto slovníky pro každý komprimovaný soubor dat dynamicky v průběhu kódování. Obecně lze říci, že adaptivní algoritmy platí za svou přizpůsobivost a větší šíři použití menší rychlostí ve srovnání se specializovanými neadaptivními algoritmy. Příkladem adaptivního kompresního algoritmu je LZW algoritmus. [5]

Semiadaptivní algoritmus je kombinací adaptivního a neadaptivního algoritmu. Tato metoda provede úvodní průchod dat kvůli vybudování slovníku a poté provede druhý průchod, při kterém se uskuteční vlastní kódování. Při použití této metody dojde k vybudování optimálního slovníku ještě před tím, než dojde ke kódování. [3]

## 2.2 Hlavní parametry výkonu kompresních algoritmů

Hlavními parametry výkonu kompresních algoritmů jsou podle [5]:

- Kompresní poměr nebo podle [8] a [9] je možné také udávat Faktor komprese
- Doba komprese
- Doba dekomprese
- Poměr mezi dobou komprese a kompresním poměrem

Kompresní poměr bývá udáván jako:

- Poměr mezi velikostí komprimovaných a nekomprimovaných dat. Podle tohoto schématu pokud komprimační program zkomprimuje soubor o původní délce 200 kB na 50 kB, je udáván kompresní poměr 1:4, popř. 25 %. Čím menší číselné vyjádření, tím lepší výsledek komprese. [5]
- Počet bitů na Byte, tj. kolik je při kompresi v průměru zapotřebí bitů pro uložení jednoho Bytu výchozího souboru. Předcházející příklad komprese souboru o původní délce 200 kB na 50 kB v komprimovaném stavu by byl tedy charakterizován kompresním poměrem 2 bpB - podle vzorce:

$$\frac{l_c}{l_u} \cdot 8$$

$l_c$  - velikost komprimovaného souboru

$l_u$  - velikost nekomprimovaného souboru

8 - počet bitů v Bytu

Samotný výpočet tedy vypadá následovně:  $\frac{l_c}{l_u} \cdot 8 = \frac{50}{200} \cdot 8 = \frac{1}{4} \cdot 8 = 2$  bpB. [6]

Faktor komprese bývá udáván jako poměr mezi velikostí nekomprimovaných a komprimovaných dat [8] [9]. Předcházející příklad komprese souboru o původní délce 200 kB na 50 kB v komprimovaném stavu by byl charakterizován kompresním poměrem 4:1 nebo 75 %. Čím vyšší číselné vyjádření, tím lepší výsledek komprese. [5]

Některé publikace, jako např. [5] zaměňují pojmy kompresní poměr a faktor komprese - resp. faktor komprese je v této publikaci vnímán jako další možnost vyjádření kompresního poměru a tento samotný výraz (tj. „Faktor komprese“) se zde nevyskytuje.

Doba dekomprese, tedy čas potřebný k rozbalení komprimovaného souboru do původní podoby, je ze všech zmíněných parametrů asi nejméně důležitý. Doba potřebná pro dekompresi bývá téměř u všech komprimačních programů kratší než doba potřebná ke kompresi, a proto tento parametr nebývá tím omezujícím nebo kritickým faktorem při výběru vhodného kompresního algoritmu.

Poměr mezi dobou komprese a kompresním poměrem často nebývá v testech kompresních algoritmů vůbec zmiňován. Vychází z požadavku dosažení co nejlepšího kompresního poměru za „přiměřený“ čas.

Při testování rychlosti algoritmu je nutné brát v úvahu parametry PC, a to především:

- Informace o procesoru (typ, taktování, počet jader, technologie výroby, materiál apod.)
- Velikost a přístupovou dobu paměti RAM
- Přístupovou dobu, rychlost čtení a zápisu pevného disku
- Stav obsazení pevného disku a jeho fragmentaci
- Použití nebo zakázání vyrovnávací paměti CACHE
- Typ operačního systému, pod kterým byly kompresní algoritmy testovány

Pokud má být test kompresních algoritmů opravdu použitelný, měl by kromě názvu těchto testovaných algoritmů obsahovat minimálně následující údaje:

- Typy souborů, na kterých bylo testování prováděno
- Nekomprimovanou velikost původního souboru
- Komprimovanou velikost souboru
- Typ hardware (především procesoru), na kterém byly testy prováděny
- Operační systém, ve kterém byl test proveden

### 3 NEZTRÁTOVÉ KOMPRESNÍ ALGORITMY

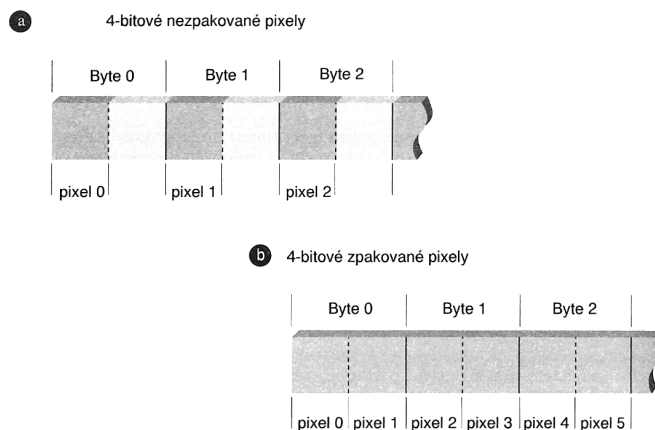
Rastrové obrazy se vyznačují vysokou paměťovou náročností, která roste kvadraticky s jejich rozlišením. Na rozdíl od komprese obecných souborů lze vycházet z vlastností a charakteristických rysů konkrétního rastrového obrazu. [1]

#### 3.1 Pixelové zhušťování

Pixelové zhušťování není typická metoda datové komprese - v podstatě ji ani nelze za kompresi označit. Je to účinný způsob ukládání dat v soustředných Bytech v paměti.

Pokud jsou v obrázkových datech obsaženy čtyři bity na pixel, tak je velmi pohodlné ukládat každý pixel do Bytu, protože Byte je nejmenší adresovatelná část paměti na většině počítačových systémů. Použitím této metody je však polovina Bytu úplně nevyužitá (viz *Obr.8-a*). Data předlohy, která obsahuje 4096 čtyřbitových pixelů tak požadují 4096 Bytů paměti. [3]

Paměť lze ušetřit tím, že namísto ukládání jednoho 4-bitového pixelu na jeden Byte, se do jednoho Bytu uloží dva takové 4-bitové pixely (viz *Obr.8-b*). Velikost paměti, do které se má uložit 4096 4-bitových pixelů potom bude mít velikost 2048 Bytů, tedy přesnou polovinu. [3]



*Obr.8 Schéma pixelového zhušťování [3]*

Paměťové založený zobrazovací hardware obvykle organizuje data předlohy jako pole Bytů, z nichž každý nese informaci o maximálně jednom pixelu. Bylo by tedy rychlejší

ukládat pouze jeden 4-bitový pixel na Byte a číst tato data přímo do paměti v řádném formátu. Ukládání dvou 4-bitových pixelů do jednoho Bytu totiž vyžaduje maskování a posouvání každého Bytu. Je zde tedy nutná volba, zda vybrat rychlejší čtení a zápis nebo redukovanou velikost souboru. [3]

### 3.2 RLE

Algoritmus Run-Length Encoding (RLE) je do češtiny překládaný jako proudové kódování [5]. Základním principem komprese je to, že se zapíše nejprve počet opakujících se totožných hodnot a poté hodnota samotná [1]. Řetězec opakujících se znaků se nazývá proud. Tento proud znaků je vždy zkomprimován do formy jednoho paketu RLE. [5]

Výhodou tohoto řešení je jednoduchost algoritmu, snadné použití a vysoká kompresní i dekompresní rychlost. Nevýhodou úzká oblast dat, na kterých tato metoda dosahuje dobré kompresní poměry. [5]

Pokud soubor obsahuje následující sekvenci znaků,

AAAAhhOOOOOj

zakóduje se tato sekvence do podoby:

3A1h4O0j

V udané posloupnosti znaků rozpozná kompresní algoritmus čtyři proudy a přiřadí jim čtyři pakety. Tyto pakety jsou:

- 3A
- 1h
- 4O
- 0j

Prvním údajem v paketu je vždy proudové číslo (počet znaků proudu snížený o jedničku). Tento údaj je následován proudovou hodnotou znaku. [5]

Z předešlého příkladu jde vidět, že zmenšení velikosti bylo dosaženo výhradně díky proudům znaků obsahujícím více než dva znaky [5]. Pokud se u sousedních pixelů ve směru ukládání (obvykle po řádcích) často neopakují stejné hodnoty, dochází k záporné kompresi, kdy se zvýší velikost komprimovaných dat oproti jejich původní formě [2].

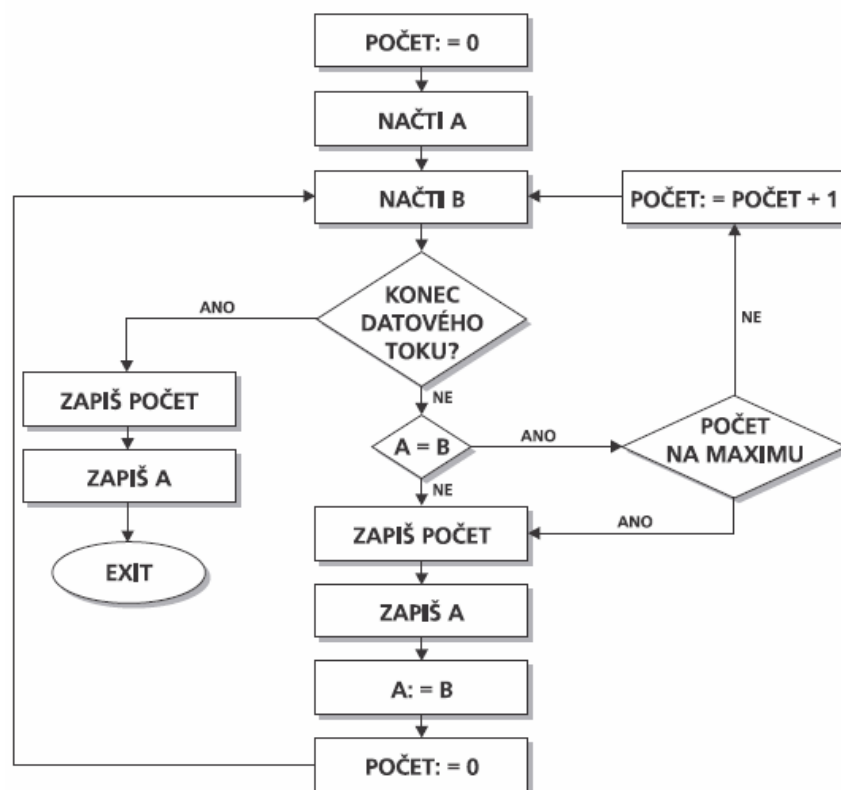
Kódování RLE je proto vhodné pro obrázky kreslené v grafickém editoru nebo pro tzv. cartoons - ilustrace s většími stejnobarevnými plochami. Také obrázek tvořený mnoha vodorovnými čarami je kódován velmi efektivně, ale tentýž obrázek otočený o 90° je zapsán téměř beze změny. [1]

RLE najde největší uplatnění u komprese jednoduchých obrázků s malou barevnou hloubkou (maximálně 256 barev). U těchto obrázků je možné dosáhnout kompresního poměru až 30 % (odpovídá kompresi dat na 30 % původní velikosti). [5]

RLE se nepoužívá pro kódování pixelů definovaných přímo hodnotami RGB, kdy není zajištěna fyzická sousednost Bytů opakujících se pixelů. Výjimkou je kódování po jednotlivých barevných rovinách. [1]

Metoda RLE není vhodná pro kompresi textových a většiny binárních souborů a také pro kompresi fotografií [5]. Důvodem menší účinnosti u fotografií je velká složitost předlohy, která obsahuje velké množství různých barev [3].

Základní struktura algoritmu RLE je zobrazena na Obr.9. [5]

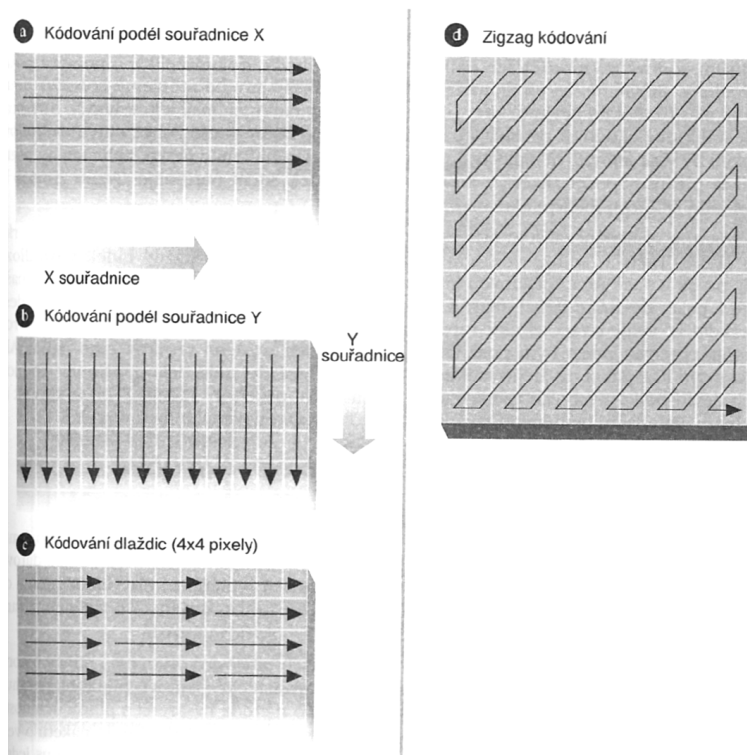


Obr.9 Základní schéma algoritmu RLE [5]

Existuje nepřehledné množství variant RLE kódování. Data předlohy jsou obvykle proudově kódována sekvenčním postupem, který bere data jako jednorozměrný proud, a ne jako dvourozměrnou mapu dat. Bitmapa se začíná kódovat od levého horního rohu a pokračuje se zleva doprava po směru vzorkovacího řádku shora dolů až do pravého dolního rohu bitmapy (*Obr.10-a*). Jiná RLE schémata však také kódují data shora dolů podél sloupců (*Obr.10-b*), kódují data do dvourozměrných dlaždic (*Obr.10-c*) nebo kódují systémem cikcak po diagonále (*Obr.10-d*). Takové zvláštní varianty jako posledně jmenovaná jsou používány jen ve speciálních aplikacích, ale jsou také dost málo rozšířené. [3]

RLE kodér by měl vždy zastavit na konci každého vzorkovaného řádku bitmapy, který má být kódován. To má některé výhody - je potřeba jen minimální velikost bufferu a je možné se vyhnout problému zvanému křížové kódování. [3]

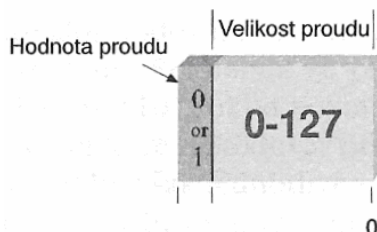
Křížové kódování je proces slučování vzorkových řádků. Vzniká tehdy, pokud kódování ztratí přehled o tom, kde začínají a končí původní řádky. Křížové kódování ušetří pár Bytů datové komprese, ale komplikuje zároveň dekódovací proces a je také časově náročné. [3]



*Obr.10 Varianty proudového kódování [3]*

### 3.2.1 Bitová úroveň kódování

Bitová úroveň kódování metodou RLE rozeznává pouze dva proudy znaků - proud jedniček a proud nul [5] - a ignoruje zarovnání na Byte nebo na slovo [3].



*Obr.11 RLE paket na bitové úrovni [3]*

Jedná se o jediný Byte logicky rozdělený do dvou částí (viz *Obr.11*). Nejvýznamnější bit každého Bytu určuje proudovou hodnotu (tj. 1 nebo 0) a sedm méně významných bitů představuje proudové číslo udávající počet opakování proudové hodnoty snížený o 1. Je zřejmé, že délka proudu musí být vždy v rozsahu 1-128 znaků, neboť do sedmi bitů lze zapsat číslo 0-127. To je také důvod, proč se jako proudové číslo používá počet znaků proudu snížený o jedničku a nikoli přímo počet znaků proudu. Pokud původní nekomprimovaná data obsahují proud delší než 128 znaků, musí být tento proud při kódování rozdělen na dva nebo více proudů. [5]

### 3.2.2 Bytová úroveň kódování

Tato modifikace kóduje proudy opakujících se Bytových hodnot a naopak si nevšímá dělení na bity nebo hranic šestnáctibitových (případně 32bitových) slov. [5]

Paket RLE se v tomto případě skládá ze dvou Bytů. První Byte udává proudové číslo v rozsahu 0-255 a druhý Byte proudovou hodnotu. Její rozsah je samozřejmě opět 0-255. Zde tedy může velikost jediného proudu, kterému bude odpovídat jeden RLE paket, dosahovat délky až 256 znaků. [5]

1	čítač	hodnota	hodnota se opakuje $(1 + \text{čítač}) \times$
0	hodnota		přímý zápis jediné 7-bitové neopakující se hodnoty
10000000		hodnota	zápis neopakující se hodnoty větší než binárně 10000000

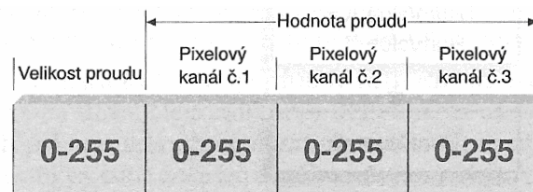
Obr.12 RLE paket na Bytové úrovni [1]

Při kódování pixelů definovaných jedním Bytem rozlišíme příznak opakování hodnotou nejvyššího bitu (viz *Obr.12*) [1]. Při nastavení nejvýznamnějšího bitu prvního Bytu na hodnotu 0 následuje za proudovým číslem přesný proud znaků, který se čte v té podobě, v jaké je zapsán. Jedná se o tzv. přímý paket. Používáním přímých paketů se částečně zamezí záporné kompresi. [5]

Efektivita komprese klesá při zpracování samostatně se vyskytujících hodnot s nenulovým nejvyšším bitem. Pokud zapisujeme obrázky definované pomocí palety, je vhodné uspořádat paletu tak, aby méně používané odstíny byly umístěny na konci palety. [1]

### 3.2.3 Pixelová úroveň kódování

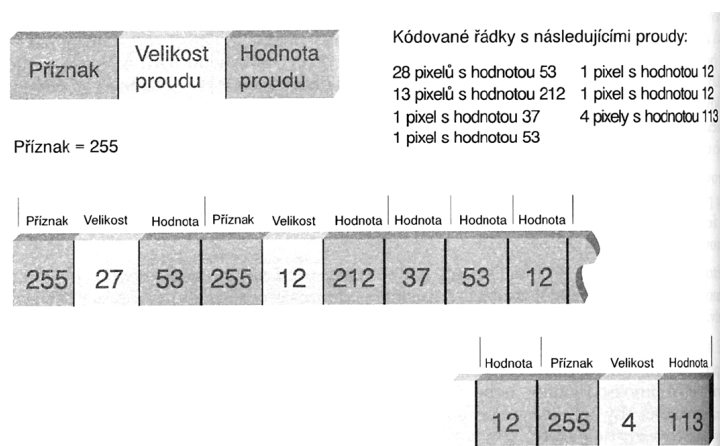
Modifikace RLE na pixelové úrovni (*Obr.13*) se používá u složitějších obrázků, kdy je jediný pixel reprezentován více než jedním Bytem. Velikost paketu RLE se v tomto případě liší podle počtu Bytů na pixel. Pokud např. obrázek podporuje schéma dat 3 Byty na pixel, bude mít paket RLE čtyři Byty - první Byte jako proudové číslo a následující tři Byty jediné proudové hodnoty. Kódovací metoda zůstává stejná jako u úrovně s jedním Bytem. [5]



Obr.13 RLE paket na pixelové úrovni [3]

### 3.2.4 Tříbytové kódování

Základní myšlenkou další metody je použití tří Bytů jako jediného paketu RLE. První Byte v tomto případě obsahuje příznakovou hodnotu, následující dva Byty tvoří klasický paket RLE. Druhý Byte opět představuje proudové číslo a třetí proudovou hodnotu. Kódování dat předlohy probíhá tak, že pokud komprimační program zjistí jedno-, dvou- nebo tří- Bytový proud, zapíše se tyto hodnoty do komprimovaného toku dat přímo (viz *Obr.14*). [5]



*Obr.14 RLE paket se třemi Byty [3]*

Při dekompresi se nejprve posoudí, zda první Byte paketu RLE obsahuje příznakovou hodnotu nebo nikoli. Pokud se jedná o příznak, expanduje se paket podle údajů proudového čísla a proudové hodnoty. Nejedná-li se o příznak, přidá se tento Byte přímo do nekomprimované podoby souboru. [5]

K nevýhodám tohoto přístupu patří to, že minimální velikost proudu dat pro „klasický“ paket RLE se zvýší na čtyři Byty, což může negativně ovlivnit kompresní poměr některých dat. Další problém nastává, pokud nekódovaný tok dat obsahuje hodnotu, která se shoduje s hodnotou vyhrazenou pro příznak. Každý takový znak musí být kódován do paketu se třemi Byty s proudovým číslem 0 (odpovídající jedinému znaku). [5]

S odkazem na předchozí odstavec podle [5] str.28: „Jako příznakovou hodnotu je tudíž nutné zvolit znak s nejmenší pravděpodobností výskytu v datech předlohy, aby v důsledku jeho častého výskytu nedocházelo ke zhoršení kompresního poměru“. Ovšem toto není zcela přesné - znak musí mít nejmenší pravděpodobnost výskytu samostatně a zároveň

nejmenší pravděpodobnost výskytu přesně  $2\times$  za sebou. Když totiž data předlohy obsahují např. 5 hodnot čísla 2, které se vždy vyskytují samostatně, číslo 8 opakující se  $10\times$  za sebou a dále všechny ostatní možné hodnoty, které jsou samostatně obsaženy častěji než hodnota 2, bylo by podle výše uvedené citované věty vhodné použít právě číslo 2 pro příznak opakování. To ovšem zapříčiní, že se těchto 5 hodnot čísla 2 uloží do 15 ( $5\times$  ve formátu 2 0 2) a ne do pouhých 5 Bytů - pro opakování čísla 8 by se použily 3 B (ve formátu 2 9 8). Pokud se však jako příznak použije číslo 8, tak se číslo 2 zakóduje do 5 Bytů a pro zápis opakování čísla 8 se použijí 3 B (ve formátu 8 9 8) - výsledek tedy bude o 10 Bytů lepší. Podobně nepřesná formulace jako výše citovaná věta se vyskytuje i v [3] str.148: „RLE algoritmus musí proto používat takovou hodnotu příznaku, která se jen výjimečně vyskytuje v nekomprimovaném toku dat“.

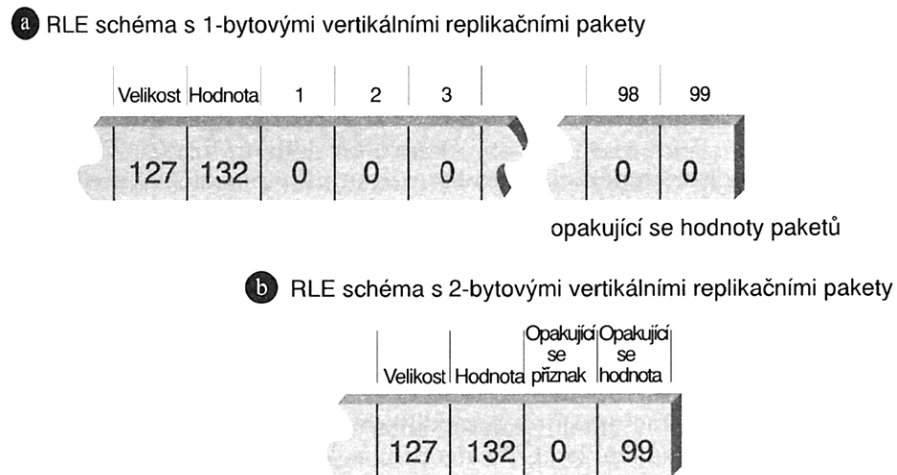
### 3.2.5 Vertikální replikační pakety

Tato modifikace komprese RLE představuje forma tzv. vertikálních replikačních paketů neboli paketů s opakovanými vzorkovými řádky. Zlepšeného kompresního poměru se dosahuje vyjádřením toho, zda se opakuje celý předchozí řádek. Je zřejmé, že se tento druh kódování hodí pouze na ty druhy dat, kde se opakování celých řádků předpokládá. [5]

Např. jeden řádek obrázku je složen z dat uložených do 512 Bytů a všechny pixely v něm jsou stejné. Za předpokladu maximálního kódovatelného proudu 128 Bytů při schématu paketu RLE se dvěma Byty se tento řádek zakóduje do osmi Bytů. Pokud má stejnou barvu prvních 100 řádků předlohy, zakóduje se při použití metody RLE na Bytové úrovni (tak jak je popsána výše) těchto 100 řádků do 800 Bytů. S metodou vertikálních replikačních paketů stačí na druhý až stý řádek jediný paket. Celkově tedy 10 Bytů. [5]

Naneštěstí, definice vertikálních replikačních paketů jsou aplikačně závislé. Dva běžně rozšířené formáty WordPerfect Graphics Metafile (WPG) a GEM Raster (IMG) používají uvedenou metodu. WPG používá jednoduché paketové schéma se dvěma Byty. Pokud je první Byte RLE paketu 0, potom jde o vertikální replikační paket. Byte, který následuje, označuje, kolikrát se musí opakovat předešlý řádek (mínus jedna). [3]

GEM Raster formát je o něco komplikovanější. Bytová sekvence 00h 00h FFh se musí objevit na začátku kódovaného řádku, aby bylo zřejmé, že jde o vertikální replikační paket. Byte, který za touto sekvencí následuje udává, kolikrát se musí opakovat předešlý řádek (mínus jedna). [3]



Obr.15 RLE schémata s vertikálními replikačními pakety [3]

### 3.3 LZ77

Abraham Lempel a Jakob Ziv vytvořili v roce 1977 to, co dnes už nazýváme rodina LZ substitučních komprimátorů [3] - jedná se o substituční (adaptivní slovníkové) metody [5].

Kompresní část algoritmu LZ77 funguje tak, že se pokouší vyhledat co nejdelší opakující se posloupnosti znaků. Pokud takovou opakující se posloupnost nalezne, zapíše na výstup pouze odkaz na předcházející výskyt řetězce. [5]

Například vstupní řetězec „leze po železe“ se zakóduje do podoby „leze po že[10,4]“. Znaky [10,4] je třeba považovat za schématicky zapsaný offset udávající, že dekodér má z předcházejících deseti znaků vybrat první čtyři. [5]

Podstatou tohoto typu komprese je tzv. posuvné okno (sliding window), které obsahuje koncovou část již přečteného (a zkomprimovaného) textu. V tomto okně se kompresní algoritmus snaží nalézt co nejdelší podřetězec odpovídající řetězci na vstupu. Pokud se to podaří, zakóduje jej v podobě odkazu na tento výskyt. Odkaz musí obsahovat ukazatel na začátek podřetězce a jeho délku. [5] [9]

Posuvné okno musí být dostatečně veliké, aby kompresní algoritmus dosáhl dobrého kompresního poměru, ale zároveň ne tak velké, aby hledání nejdelšího řetězce v tomto okně neúměrně prodloužilo dobu komprese. Ve většině případů se používá velikost okna v řádu kB. [5] [9]

Posuvné okno obsahuje dvě části: prohlížečí okno a aktuální okno (look ahead buffer). Na začátku algoritmus nastaví posuvné okno tak, aby začátek vstupu obsahovalo aktuální okno. [5]

Potom pokaždé v posuvném okně najdeme co nejdelší počáteční podřetězec (předponu) řetězce z aktuálního okna začínající v prohlížečím okně. Tento podřetězec se pak zakóduje v podobě ukazatele na počátek podřetězce v prohlížečím okně a jeho délku. [5]

Stručný nástin algoritmu popsaného v předcházejícím odstavci: [5]

```
1 while (aktuální okno není prázdné)
2 {
3   najdi informaci (ukazatel, délka) nejdelší předpony aktuálního okna
4   if (délka > minimální délka)
5   {
6       write (ukazatel, délka)
7       posuň posuvné okno o délku znaků doleva
8   }
9   else
10  {
11      write (první znak aktuálního okna)
12      posuň posuvné okno o 1 znak doleva
13  }
14 }
```

Existuje několik způsobů jak rozeznat dvojici znaků nesoucí informaci o dvojici (ukazatel, délka) od jednotlivých znaků na výstupu. Nejpoužívanějším je zápis LZSS (Storer, Szymanski): <1, ukazatel, délka> nebo <0, jednotlivý znak>. Nula na počátku signalizuje jednotlivý znak, jednička informační dvojici znaků. [5]

Dekomprese souboru zkomprimovaného metodou LZ77 je velice jednoduchá a rychlá. Vždy, když dekompresní algoritmus narazí na offset udávající ukazatel a délku řetězce, tak tento řetězec zkomprimuje na výstup. [5]

### 3.4 LZW

LZW (Lempel-Ziv-Welchova) komprese je jednou z nejrozšířenějších metod používaných v počítačové grafice. V roce 1984 Terry Welch při práci pro Unisys modifikoval LZ78 komprimátor pro vysoce výkonné diskové kontroléry - výsledkem byl LZW algoritmus. [3] Tento způsob komprese je rozšířen při bezztrátové kompresi grafických, binárních i textových dat. Používá se např. v grafických formátech GIF, TIFF, programech jako jsou ARJ nebo PKZIP nebo také jako součást modemových standardů pro přenos dat. [5] LZW je tedy všeobecný kompresní algoritmus, který je schopen zpracovávat různé typy dat [3].

Tato komprese má velmi dobrý kompresní poměr, rychlou kompresi i dekompresi, malé nároky na paměť [5] a pracuje s celými čísly [3]. Další výhodou je adaptivní metoda vytvářející dynamický substituční slovník. Tento slovník není nutné ukládat pro potřeby dekomprese - dekompresní část algoritmu vytvoří duplicitní slovník - a tím je zmenšena velikost výsledného souboru. [5] LZW také komprimuje data do Bytů a ne do slov, a proto může být kódovaný výstup jak v systému velký endián, tak v systému malý endián. Na problémy bitového a výplňového uspořádání však lze narazit stejně. [3]

Za nevýhodu se dá považovat rychlý nárůst slovníkových kódů odkazujících na řetězce původního souboru, což vede v některých případech k zaplnění paměti určené pro slovníkové kódy. Z toho pak vyplývá smazání aktuálního slovníku a jeho nové vytváření bez možnosti použití smazaných odkazů. V případě větších slovníků zase narůstá doba vyhledání řetězce ve slovníku. [5]

Základním principem je vyhledávání stejných posloupností Bytů v originálním souboru. Pomocí odkazů na tyto posloupnosti dat algoritmus buduje datový slovník. Každý další výskyt takové posloupnosti se zakóduje buď jako odkaz na předcházející výskyt posloupnosti nebo slovníkový odkaz spojený s příslušným řetězcem. [5] Datové vzorky (podřetězce) jsou definovány jako toky dat a shodují se se vstupy do slovníku [3].

Kompresní algoritmus postupně rozpoznává a ukládá do slovníku řetězce znaků. Tyto řetězce nahrazuje ve výstupním textu celými čísly z předem definovaného intervalu. Definice intervalu je závislá na charakteru komprimovaných dat. Například při kódování řetězce znaků zobrazených v osmibitovém zobrazení (znaků ASCII) je prvních 255 čísel vyhrazeno pro zobrazení samostatných znaků z původního souboru. Čísla nad 255 se pak

přidělují jednotlivým nalezeným řetězcům. Přitom se vytváří slovník již rozeznáných řetězců, který se v průběhu komprese udržuje v paměti počítače. [5]

Běh algoritmu LZW začíná s prázdným slovníkem a řetězcem S (slovo - word) obsahujícím první znak zdrojového souboru. Vždy po přečtení dalšího znaku Z zjistí, jestli se řetězec S+Z vyskytuje ve slovníku. Pokud ano, pouze prodlouží řetězec S o znak Z, jinak zapíše nový odkaz na řetězec do slovníku. Pokud řetězec S obsahuje jediný znak, bude do slovníku zanesen pouze jediný znak. Vzhledem k tomu, že do slovníku se zapisují čísla větší než 255, je nutné i tento jediný znak zapsat v příslušné podobě. K vyjádření odkazu na řetězec se většinou používá 12bitová hodnota. V tomto případě tedy čísla 0–255 jsou jednotlivé znaky a čísla 256–4095 kódy řetězců do slovníku. [5] [9] Pro dosažení datové komprese je nutné zaručit, aby hodnota (resp. fráze) vyjadřující slovo měla fyzickou velikost menší než dané slovo [3].

Pokud se slovník zaplní dříve, než je přečten celý soubor, je celý slovník smazán a začíná se plnit znovu. Někdy se pro zlepšení účinnosti místo smazání slovníku používá algoritmus LRU (last-recently-used) pro odstranění nepoužívaných řetězců ze slovníku. [5]

Při kompresi se velmi často zjišťuje, jestli je již nějaký řetězec nebo jeho kód ve slovníku. Pro tento účel je nutné použít některou datovou strukturu pro ukládání informací s efektivní metodou vyhledávání, např. strom nebo hash-table. Následně právě popsaného algoritmu uvádějí následující řádky: [5]

```
1 set S = NULL
2 While (!eof(input))
3 {
4   přečti znak Z
5   if (SZ ve slovníku)
6     S = SZ
7   else
8     {   Zapiš kódovou hodnotu pro S
9       Přidej SZ do slovníku
10      S = Z } }
```

Pro větší názornost uvedeného algoritmu je na *Obr.16* konkrétní příklad zakódování vstupního řetězce „WEB/WEB/WEB!“. [5] [9]

<b>Řetězec W</b>	<b>Přečtený znak</b>	<b>Výstup</b>	<b>Nová položka ve slovníku</b>
		W	
W	E	W	(256) = WE
E	B	E	(257) = EB
B	/	B	(258) = B/
/	W	/	(259) = /W
W	E		
WE	B	(256)	(260) = WEB
B	/		
B/	W	(258)	(261) = B/W
W	E		
WE	B		
WEB	!	(260)	(262) = WEB!
!	(eof)	!	

*Obr.16* Příklad postupu algoritmu LZW při kódování [5]

Dekompresní část algoritmu postupně čte kódy komprimovaného souboru, zapisuje příslušející řetězce na výstup a přidává nové řetězce do slovníku. Do slovníku je vždy přidán řetězec reprezentovaný předcházejícím kódem a první znak z řetězce s aktuálním kódem. Pro případ, že by byl přečten kód, kterému ještě nebyl přiřazen řetězec (tato situace může nastat např. pokud původní text obsahoval několik stejných znaků za sebou), je do slovníku přidán řetězec skládající se z předcházejícího řetězce a jeho posledního znaku. Takto vytvořený slovník bude totožný s tím, který vytvořil kompresní algoritmus - pro potřeby dekomprese tedy není nutné uchovávat datový slovník vytvářený při kompresi. [5] [9]

Dekompresní schéma algoritmu LZW lze zapsat následujícím způsobem: [5]

```
1  čti prom
2  zapiš prom
3  while (!eof(input))
4  {
5    čti code
6    zapiš řetězec odpovídající kódu code (tj. table[code]) - to může
   být přímo znak code
7    přidej řetězec table[last] + první znak table[code] do slovníku
8    prom = code
9  }
```

Předlohy plné šumu mohou významně degradovat úroveň LZW komprese. Doporučuje se odstranit šum z předlohy (zpravidla vynulováním posledních dvou nejméně významných bitů - tím ovšem vzniká ztráta informace), což může výrazně zvýšit kompresní účinnost. Jinými slovy, pokud data nejsou komprimována dobře v jedné formě, lze je převést do formy jiné, která půjde zkomprimovat lépe. [3]

### 3.5 Huffmanovo kódování

Huffmanovo kódování patří do skupiny algoritmů, které pracují na základě různých pravděpodobností znaků kódovaných dat. Z této věty je zřejmé, že Huffmanovo kódování je určeno především pro komprimaci textových souborů. [5] Huffmanova konstrukce minimálního kódu je vždy optimální, ale není jednoznačná [9].

Při kompresi se postupuje tak, že nejprve komprimační algoritmus zjistí pravděpodobnosti výskytů jednotlivých znaků (popř. jejich kombinací) a každému znaku (kombinaci znaků) přiřadí jedinečný kód. Takovéto kódy se liší svou bitovou délkou. Tato část algoritmu je nejdůležitější a musí být navržena tak, aby přiřazení kódů znakům respektovalo požadavek na přiřazení bitově nejkratších kódů znakům s častějším výskytem a bitově delších kódů znakům s méně častým výskytem. Pak algoritmus postupně načítá znaky vstupního souboru, nachází odpovídající předem přiřazené kódy a tyto kódy zapisuje na výstup. [5]

Např. soubor obsahuje pouze znaky A, 8, 0, K, R a pravděpodobnosti výskytu těchto znaků v souboru určeném pro kompresi s následujícími hodnotami: [5]

znak A se v souboru vyskytuje s pravděpodobností 50 %,

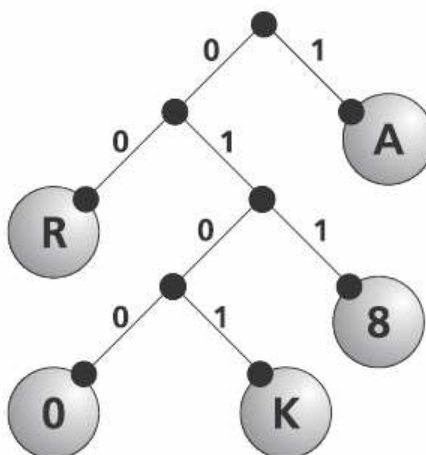
znak 8 se v souboru vyskytuje s pravděpodobností 12,5 %,

znak 0 se v souboru vyskytuje s pravděpodobností 6,25 %,

znak K se v souboru vyskytuje s pravděpodobností 6,25 %,

znak R se v souboru vyskytuje s pravděpodobností 25 %.

Binární strom se podle zásad Huffmanova algoritmu tvoří od koncových „listů“ stromu, tj. od znaků s nejmenší pravděpodobností. Nejprve dojde k výběru znaků s nejmenší pravděpodobností výskytu a z nich se vytvoří dva koncové listy stromu. Potom se vybere znak s nejnižší pravděpodobností z dosud nezpracovaných znaků, a tak se postupuje až ke kořeni stromu (viz *Obr.17*). [5]



*Obr.17 Příklad binárního stromu  
Huffmanova kódování [5]*

Každý uzel binárního stromu představuje součet pravděpodobností všech „listů“, které jsou ve stromové struktuře pod ním [5].

Vytvořením binárního stromu jsou definovány kódy přiřazené podle pravděpodobnosti výskytu jednotlivým znakům souboru. Tyto kódy se přiřadí jednotlivým znakům tak, že ke každému znaku se postupuje stromovou strukturou. Při pohybu od uzlu k uzlu zaznamenáme jedničku vždy při cestě vpravo dolů a nulu při cestě vlevo dolů. [5]

Pokud se budeme držet binárního stromu vytvořeného na *Obr.17*, budou kódy přiřazené jednotlivým znakům vypadat takto: [5]

A: 1

8: 011

0: 0100

K: 0101

R: 00

Binárních stromů lze uvedeným způsobem vytvořit více, avšak celková komprimovaná délka souboru bude vždy stejná, jelikož počet bitů přiřazených jednotlivým znakům bude ve všech případech stejný. [5]

Komprese na základě vytvořeného binárního stromu probíhá tak, že kompresní algoritmus čte sekvenčně znaky ze vstupního souboru a do výstupního pak zapisuje jim příslušné sekvence bitů [5].

Dekomprese vyžaduje znalost původního binárního stromu. Pokud jsou kódy ve zkomprimované podobě souboru uloženy bezprostředně za sebou, bude již dekompresní algoritmus schopen rozpoznat znak příslušející ke kódu. Algoritmus čte kód (o kterém ještě není známo, jak je dlouhý) bit po bitu a podle hodnoty bitu postupuje po binárním stromě od kořene až k „listu“ stromu představujícímu znak. Tento znak je již původním znakem, který byl při kompresi zakódován. [5]

Slabinou tohoto algoritmu je mimo jiné i to, že binární strom lze bez problémů sestavit pouze v případě, že pravděpodobnosti výskytu všech znaků vstupního souboru jsou mocninou čísla  $1/2$ . Pouze v tom případě každé vyšší patro vytvářeného stromu obsahuje uzel nebo list stromu s pravděpodobností výskytu dvojnásobnou oproti listům či uzlům ležícím o patro níž. Je zřejmé, že ideálního kompresního poměru může být dosaženo pouze při dodržení této zásady. V praxi však s něčím takovým nelze počítat, a proto je zapotřebí zaokrouhlovat pravděpodobnosti výskytu jednotlivých znaků. [5]

Zaokrouhlování ovšem musí být prováděno při dodržení dvou základních zásad: [5]

- součet všech procentuálních hodnot musí dávat hodnotu 100 %;
- je nutné dbát na to, že v každém patře vytvářeného stromu může být maximálně určitý počet listů a uzlů. Tento počet je dále omezen již vytvořenými patry stromu.

Složitosti vznikající při dodržování poslední zásady lze demonstrovat na následujícím příkladu: [5]

znak A se v souboru vyskytuje s pravděpodobností 46 %,

znak 8 se v souboru vyskytuje s pravděpodobností 22 %,

znak 0 se v souboru vyskytuje s pravděpodobností 5 %,

znak K se v souboru vyskytuje s pravděpodobností 5 %,

znak R se v souboru vyskytuje s pravděpodobností 22 %.

Při vytváření binárního stromu podle zásad Huffmanova algoritmu se naleznou nejprve dva znaky s nejnižší pravděpodobností (0 a K). Jejich pravděpodobnosti zaokrouhlíme na 6,25 %, což je mocnina  $\frac{1}{2}$ . Následují znaky 8 a R se stejnou pravděpodobností. Zaokrouhlení obou příslušných pravděpodobností na 25 % je nepřípustné, protože na jediný zbývajících znak by zbyla pravděpodobnost 37,5 % - ta je ovšem v binárním stromě nežádoucí. Je tedy nutné pravděpodobnost jednoho z těchto znaků zaokrouhlit na 12,5 % a pravděpodobnost druhého zaokrouhlit na 25 %. To znamená, že ačkoli se oba znaky ve vstupním souboru vyskytují se stejnou pravděpodobností, jeden z nich bude v komprimované formě reprezentován třemi bity a druhý dvěma bity. [5]

Tento jednoduchý příklad nedemonstruje všechny problémy spojené s tvorbou binárního stromu, ale nastiňuje příčinu, proč se do kompresního schématu zavádějí nežádoucí nepřesnosti, které zhoršují výsledný kompresní poměr. Zároveň se algoritmus tvorby binárního stromu stává složitější. [5]

### 3.5.1 CCITT kódování

CCITT (International Telegraph and Telephone Consultative Committee) je standardizační organizace, která vyvinula sérii komunikačních protokolů pro přenos černobílých předloh přes telefonní linky a datové sítě. Tyto protokoly jsou obecně známy jako CCITT standardy T.4 a T.6, ale častěji se nazývají CCITT Group 3 a Group 4 komprese. Toto kódování je speciálním typem Huffmanova kódování. [3]

Kódování a dekódování Group 3 je rychlé, dosahuje dobrého kompresního poměru (1:5 až 1:8) pro široké spektrum typů dat a obsahuje informace, které pomáhají dekodéru Group 3 v detekci a odstraňování chyb. [3]

Data zakódovaná Group 4 jsou oproti stejným datům zakódovaným Group 3 poloviční - kompresní poměr až 1:15. Ačkoli je Group 4 docela složité účinně implementovat, kóduje přibližně stejně rychle jako Group 3 a v některých implementacích je dokonce rychlejší. Group 4 byla vytvořena pro sítě dat, takže neobsahuje synchronizační kódy, které se používají pro detekci chyb. [3]

CCITT algoritmy jsou algoritmy neadaptivní - tzn. nepřizpůsobují se každé bitmapě tak, aby byla kódována s optimální účinností, ale používají pevnou tabulku kódových hodnot, které byly vybrány podle referenčního vzorku dokumentů obsahujících text i grafiku. [3]

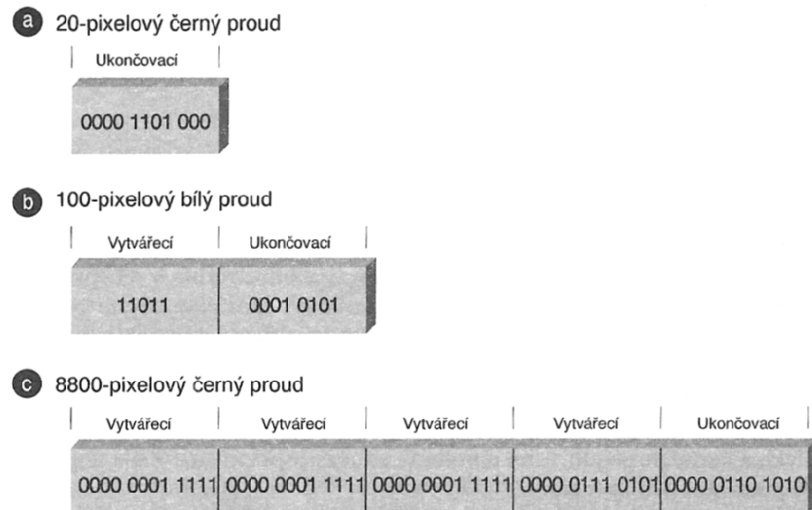
Protože byl CCITT algoritmus optimalizován pro strojově a ručně psané dokumenty, je zřejmé, že předlohy, které se radikálně liší ve své kompozici nebudou velmi dobře komprimovány. U mnohých z nich dojde k záporné kompresi. [3]

CCITT definuje 3 algoritmy, které se používají pro kódování dvourozměrných dat: [3]

- Group 3 jednoúrovňové (G31D)
- Group 3 dvourozměrné (G32D)
- Group 4 dvourozměrné (G42D)

U G31D se provádí komprese RLE a hodnoty čítače opakování jsou dále kódovány Huffmanovým kódováním. Používají se zde krátké kódy pro typické úseky bílých resp. černých pixelů a dále - kód FILL (stejná barva až do konce řádku), EOL (konec řádku) a RTC (konec dat). [2] Data je možné dekodovat bez vyrovnávací paměti. Všechny vzorkové řádky jsou kódovány tak, aby začínaly kódovým slovem bílého proudu. Pokud by tomu tak nebylo a řádek by začínal černým proudem, přidá se na začátek takového řádku kódové slovo bílého proudu o délce nula. [3]

Pixelové proudy s délkou 0-63 jsou kódovány jediným ukončovacím kódem. Proudů o velikosti 64-2623 jsou kódovány jedním vytvářecím a jedním ukončovacím kódem. Proudů větší než 2623 pixelů se kódují nejméně jedním vytvářecím kódem a jedním ukončovacím kódem. Délka proudu je součet hodnot délek každého kódového slova. Příklady G31D kódování je na *Obr.18*. [3]



Obr.18 Kódování CCITT G31D [3]

Ve standardu G32D se provádí kódování pozic změny barvy, relativně vůči předchozí pozici a uvažuje se i změna oproti předchozímu řádku (ve vertikálním směru). Každý K-tý řádek se kvůli spolehlivosti kóduje pomocí G31D. [2] Tento standard obsahuje kontrolní kódy pro detekci a odstranění případných chyb. [3]

Kódování G42D je kromě několika modifikací stejné jako kódování G32D. Neobsahuje však žádné kontrolní kódy, není zde totiž nutné zajistit detekci ani opravu chyb. Kódování G42D bylo vytvořeno speciálně pro kódování dat na disku a na datových sítích. [3]

### 3.6 Shannon-Fanovo kódování

Shannon-Fanovo kódování je velice podobné Huffmanovu. Rozdíl mezi oběma algoritmy spočívá v konstrukci binárního stromu. Tvorba binárního stromu v Shannon-Fanově modifikaci je poněkud jednodušší. Lze ji shrnout do dvou následujících kroků: [5]

- Rozdělení souboru symbolů na dvě skupiny se stejnou nebo co nejpodobnější celkovou pravděpodobností znaků obsažených v obou skupinách.
- Opakování prvního kroku na všechny dosud vytvořené skupiny, dokud každá skupina nebude obsahovat jediný znak.

Rozdíl mezi způsoby vytváření binárních stromů v Huffmanově a Shannon-Fanově variantě je v tom, že Huffmanovo kódování vytváří strom od koncových listů směrem ke kořenu, zatímco Shannon-Fanova metoda postupuje obráceně - od kořene k listům. [5]

Zatímco Huffmanova varianta při správném použití generuje vždy optimální kódy pro jednotlivé znaky, jednodušší Shannon-Fanova metoda může v některých případech použít několik bitů navíc. [5] [9]

Např. soubor obsahuje pouze znaky A, 8, 0, K, R a pravděpodobnosti výskytu těchto znaků v souboru určeném pro kompresi s následujícími hodnotami: [5]

znak A se v souboru vyskytuje s pravděpodobností 50 %,

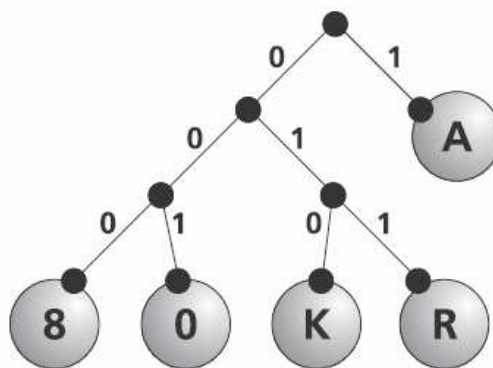
znak 8 se v souboru vyskytuje s pravděpodobností 12,5 %,

znak 0 se v souboru vyskytuje s pravděpodobností 6,25 %,

znak K se v souboru vyskytuje s pravděpodobností 6,25 %,

znak R se v souboru vyskytuje s pravděpodobností 25 %.

Vytvořený Shannon-Fanův binárního strom je možné zhlédnout na *Obr.19*. Kód příslušející ke znaku A je reprezentován jediným bitem, zatímco všem ostatním znakům je přiřazen kód tříbitový. Zanikne tak rozdíl mezi pravděpodobnostmi výskytu znaku 5 % a 22 %. [5]



*Obr.19 Příklad binárního stromu Shannon-Fanova kódování [5]*

### 3.7 Aritmetické kódování

Huffmanovo kódování dosáhne optimálních výsledků pouze v případě, kdy pravděpodobnosti jednotlivých znaků jsou celočíselnými mocninami  $1/2$ . V ostatních případech by bylo optimální na zakódování jednoho znaku použít neceločíselný počet bitů, což však binární soustava neumožňuje. Aritmetické kódování odstraňuje tento nedostatek tím, že kóduje celou zprávu jako jediné kódové slovo. [5]

Základní myšlenku použitou v této kompresní metodě lze popsat ve stručnosti takto: aritmetické kódování reprezentuje celou zprávu jako číslo z intervalu  $<0,1$ ). Na začátku kódování se uvažuje celý tento interval. Jak se zpráva prodlužuje, postupně se tento interval zužuje - přibližují se k sobě horní a dolní mez nově vytvářeného intervalu. Na konec stačí zapsat libovolné číslo z výsledného intervalu - to samo o sobě reprezentuje celou zprávu. [5]

Algoritmus komprese lze nastínit jako následující sekvenci kroků: [5]

- Zjištění pravděpodobností výskytu jednotlivých znaků ve zdrojovém souboru.
- Rozdělení intervalu  $<0,1$ ) na podintervaly, jejichž vzájemný poměr velikostí odpovídá poměru pravděpodobností jednotlivých znaků (seřazených podle abecedy).
- Uložení tohoto základního rozdělení intervalu  $<0,1$ ).
- Vlastní komprese víceznakové zprávy. Komprese víceznakové zprávy probíhá tak, že se nejprve vybere první znak vstupního souboru. Ten zúží interval  $<0,1$ ) na podinterval příslušející tomuto znaku tak, jak mu byl přidělen v druhém bodě celkového algoritmu aritmetického kódování. Tento podinterval je následně rozdělen stejným způsobem jako dříve celý interval  $<0,1$ ). Po načtení dalšího znaku je podinterval dále zúžen podle načteného znaku. Tak to půjde dále až do načtení posledního znaku zprávy. Čím je kódovaný znak pravděpodobnější, tím se interval zúží méně.
- Posledním bodem je vybrání kteréhokoli zlomku náležejícího do výsledného nejjemnějšího podintervalu a jeho převedení do binární formy [5]. Vznikne tak číslo, které přesně definuje daný interval [8].

Kromě kódové hodnoty je pro potřeby dekomprese zakódované zprávy nezbytné uložit pravděpodobnosti jednotlivých znaků a počet znaků původní zprávy. [5]

Při dekompresi je možné postupovat podle následujícího návodu: [5]

- výběr podintervalu  $\langle K(i), K(i+1) \rangle$  tak, aby kód náležel do tohoto intervalu
- zjištění a zápis znaku, který náleží svou pravděpodobností do tohoto intervalu.

Při převádění myšlenky aritmetického kódování do počítačové praxe se lze setkat s celou řadou problémů - nejzávažnější je asi potřeba konverze reálných čísel na bitové vyjádření. Navíc s každým reálným číslem rozdělujícím základní interval  $\langle 0, 1 \rangle$  je nezbytné pracovat s vyloučením jakéhokoli zaokrouhlování. [5]

Při vyjádření výsledného zlomku v bitovém vyjádření je důležitý fakt, že jmenovatel musí být mocninou čísla 2. Jedině tak je možné jednoznačně stanovit délku výsledného bitového vyjádření celého zlomku. Počet bitů bitového vyjádření je takový, aby pomocí nich bylo možné vyjádřit všechny hodnoty v intervalu  $\langle 0, \text{jmenovatel}-1 \rangle$ . V *Tab.1* je několik jednoduchých příkladů toho, jak se zlomky konvertují na svá bitová vyjádření (nejnižší bit je uveden vpravo). [5]

*Tab.1 Konverze zlomku na binární vyjádření [5]*

Zlomek	Binární vyjádření
1/2	1
1/4	01
3/4	11
5/8	101
31/32	11111

Velkým problémem je u popsaného kompresního algoritmu rychlý růst počtu platných desetinných míst v racionálních číslech, která představují dolní a horní meze intervalů. Ve většině případů již u souborů o deseti znacích roste počet platných cifer těchto dělicích bodů na dvacet až třicet číslic. Z toho je zřejmé, že celé soubory o velikosti mnoha megabajtů nelze kódovat jediným číslem, neboť počet platných míst u čísel v mezivýpočtech by rostl prakticky neomezeně. [5]

Je tedy nutné rozdělit každý soubor na jednotlivé bloky a ty komprimovat samostatně. Bohužel se ukazuje, že tyto bloky většinou nemohou přesahovat 10 bajtů. S každým blokem o deseti bajtech, který se ukládá, je spojeno několik desítek matematických operací s racionálními čísly o desetinném rozvoji jdoucím do dvaceti až třiceti platných číslic. [5]

Tyto operace musí být navíc emulované softwarově, jelikož problém si nelze ulehčit ani pomocí matematického koprocesoru, neboť ten podporuje reálná čísla vyjádřená pouze v 6 bajtech. To je pro aritmetické kódování nedostačující. Softwarová emulace způsobuje poměrně dlouhou dobu zpracování. I přes rychle se rozvíjející hardwarové možnosti nelze přehlédnout vysokou náročnost kompresního algoritmu na zdroje počítače - především na paměť a procesor. [5]

Dalším problémem tohoto druhu komprese je např. složitost způsobu zjišťování, jak dlouhé bude výsledné bitové vyjádření kódu reprezentujícího vstupní posloupnost znaků. [5]

Ani tím však problémy s praktickou realizací algoritmu nekončí. Jako obtížný se jeví již první krok celého algoritmu - zjištění pravděpodobnosti výskytu jednotlivých znaků. To lze samozřejmě řešit sekvenčním načtením celého souboru a prostým výpočtem pravděpodobností z absolutního počtu jednotlivých výskytů. U rozsáhlých souborů však dojde k nepříjemnému prodloužení doby komprese, do které je nutné čas na zjištění pravděpodobností výskytů započítat. U dlouhých souborů je doba komprese limitním faktorem již bez tohoto zdržení. [5]

Nejúčinnější metodou se dnes jeví způsob zjišťování frekvence znaků založený na markovovském zdroji dat - DMC (Dynamic Markov Coding). Má-li vstupní soubor  $N$  různých znaků, je tento typ zdroje daný tzv. počátečním vektorem pravděpodobností: [5]

$$p(0), p(1), \dots, p(N-1)$$

a maticí přechodových pravděpodobností: [5]

$$\begin{pmatrix} p(0;0) & \dots & p(0;N-1) \\ \dots & \dots & \dots \\ p(N-1;0) & \dots & p(N-1;N-1) \end{pmatrix}$$

Vektor počátečních pravděpodobností a matice přechodových pravděpodobností spolu souvisí vztahem: [5]

$$p(y) = \sum_{x=0}^{N-1} p(x, y) \cdot p(x)$$

Pravděpodobnost  $p(y, x)$  označuje pravděpodobnost, že v daném souboru se po znaku  $x$  vyskytuje znak  $y$ . Pravděpodobnost  $p(x)$  odpovídá pravděpodobnosti, že prvním znakem sekvence bude znak  $x$ . [5]

Matice přechodových pravděpodobností se upravuje a doplňuje v průběhu komprese podle výskytů jednotlivých znaků ve vstupním souboru. Jasnou výhodou tohoto přístupu je fakt, že na rozdíl od statických tabulek znaků se jedná o adaptivní variantu algoritmu, kterou lze použít i na netextová data. [5]

### 3.8 JBIG

JBIG (Joint Bi-level Image Experts Group) je metoda pro bezztrátovou kompresi obrazů, které obsahují ideálně pouze černou a bílou a využívá se především u faxů [5]. Pomocí metody JBIG však lze komprimovat i předlohy barevné či ve stupních šedi o hloubce do 256 bitů na pixel [3]. Pro obrázky s odstíny šedi používající více než 8 bitů na pixel by již tato metoda nebyla tak účinná jako metody jiné [5]. Tyto předlohy s větším počtem bitů na pixel jsou komprimovány po bitových plochách, nikoli po pixelech - např. 8 bitový obrázek komprimovaný metodou JBIG bude kódován do osmi oddělených bitových ploch. [3]

Doporučuje se, aby kvůli normalizaci změn mezi hodnotami sousedních Bytů v obrazových datech byla každá bitová plocha předzpracována algoritmem šedého kódování - tento postup zvyšuje účinnost kodéru JBIG. [3]

JBIG byla vyvinuta s cílem zcela nahradit méně účinné kompresní algoritmy MR (Modified READ) a MMR (Modified Modified READ), používané v protokolech datového přenosu Group 3 (G3) výboru CCITT respektive Group 4 (G4). [3]

JBIG dosahuje na naskenovaných předlohách s čárovými kresbami a tištěným textem o 10 až 50 % lepšího kompresního poměru než Group 4 a až o 500 % na počítačem generovaných obrazech tištěného textu. Dvourozměrné obrazy zpracované polotónováním či tónováním jsou touto metodou komprimovány na velikost 2 až 30-krát menší než kompresí Group 4. Těchto vynikajících kompresních poměrů dosahuje JBIG díky schopnosti přizpůsobit se obsahu kódovaných obrazových dat. [3]

JBIG byl zatížen celou řadou patentovaných postupů [3]. Dne 26.2.2011 vypršely ve všech zemích kromě USA veškeré patenty na algoritmy použité v JBIG. Celosvětově není komprese JBIG zatížena žádnými patenty od 4.4.2012. [10]

Před samotným kódováním je obraz rozdělen do bitových rovin, kde je v každé rovině obraz redukován na hloubku šedi jediného bitu na pixel. Pokud má již originální obrázek pouze 1 bit na pixel, fáze rozdělení na bitové roviny odpadá. [5]

Další rozdělení obrazu zajišťuje, že obraz bude k příjemci posílán po částech tzv. progresivní metodou. Obraz je rozdělen na několik horizontálních pruhů, které jsou kódovány postupně. Každý pruh se navíc vysílá ve vrstvách podle rozlišení. Přijímací zařízení bude obraz rekonstruovat tak, jak jej bude přijímat od hrubších (nejvyšších rozlišení) k jemnějším detailům (nejmenším rozlišení). Tato metoda používá postupné zdvojování úrovně rozlišení. Předloha se třemi vrstvami má tedy dvě zdvojení. Počet kódovatelných zdvojení není nijak omezen. [3] [5]

Používá se také sekvenční kódování, při kterém se naopak obraz kóduje shora dolů bez jakéhokoliv prokládání a jako jeden obrázek. Sekvenční předloha JBIG se dekóduje jedním průchodem a dosahuje přinejmenším stejně dobrého kompresního poměru jako Group 4, ovšem obecně horšího než při použití progresivní metody. [3] [5]

Sekvenční i progresivní kódování metodou JBIG jsou zcela kompatibilní. Předlohy komprimované použitím sekvenčního kódování jsou čitelné progresivními dekodéry JBIG. Sekvenční dekodéry JBIG jsou však z předloh kódovaných progresivně schopny číst pouze první vrstvu s nejnižším rozlišením. Progresivní kódování má vyšší nároky na paměť - k uložení referenčních dat musí být použita vyrovnávací obrazová paměť. [3]

Pro kompresi obrazu se používá adaptivní aritmetický kodér (Q CODER), konkrétně se jedná o bezztrátový adaptivní algoritmus podobný Huffmanovu kódování. V průběhu komprese se postupně vytváří adaptivní rozdělení pravděpodobností výskytu bílých a černých pixelů. Pro pravděpodobnější symboly se potom v kompresi použije menší počet bitů než pro symboly méně pravděpodobné. [3] [5]

Dvouúrovňové předlohy se skládají pouze ze dvou barev a k uložení každého pixelu slouží jediný bit. Nejobvyklejším typem dvouúrovňových předloh jsou černobílé dokumenty. Hodnotami 0 a 1 jednoho bitu však mohou být reprezentovány jakékoli dvojice barev - barva popředí a barva pozadí. [3]

U běžných dvouúrovňových kompresních algoritmů probíhá kódování po jednom řádku, a to metodou proudové délky. Algoritmy tohoto druhu bývají označovány jako kódovací metody 1D. U metod 2D dochází k zakódování pixelových proudů popisem rozdílu mezi hodnotami pixelu v aktuálním a předchozím řádku. [3]

Nadbytečná obrazová data jsou u metody JBIG kódována porovnáním pixelu ve vzorkované řádce s množinou pixelu, které už kodér vzorkoval. Těmto dodatečným pixelům se říká šablona, a vytvářejí jednoduchou mapu vzoru složeného z pixelu, které obklopují právě kódovaný pixel. Z hodnot těchto pixelů se určí nadbytečné vzory v obrazových datech. Tyto vzory jsou následně komprimovány adaptivním aritmetickým kompresním kóděrem. [3]

### 3.9 JBIG 2

Navrhovaným cílem pro JBIG2 bylo dosáhnout lepší bezztrátové komprese, než kterého dosahují jiné již existující standardy a zároveň umožnit při ztrátové kompresi mnohem vyšší kompresní poměr při téměř nezatelném snížení kvality [11]. JBIG2 dosahuje 3 až 5-krát lepší kompresi než Group 4, a také 2 až 4-krát lepší kompresi než JBIG. Obrázky mohou být dekomprimovány s rychlostí dekódování až 250 millionů pixelů za sekundu. [12]

JBIG2 umožňuje využít Huffmanova kódování se standardními či předdefinovanými tabulkami. Kódová tabulka se skládá z řádku, kde každý řádek říká, jak zakódovat konkrétní hodnotu nebo jak zakódovat hodnotu z určitého rozsahu. U faxu JBIG2 využívá také variantu nazývanou modifikované Huffmanovo kódování. Je určené pro kódování černobílých obrázků. Kombinuje variabilní délku kódu Huffmanova kódování s kódováním opakujících se dat v RLE (run-length encoding). [11]

V JBIG2 je možné využít adaptivní verzi aritmetického kódování a její varianty. Jednou z variant je adaptivní binární aritmetické kódování. Jelikož kóduje nuly a jedničky, dělí interval na dva podintervaly. Jejich šířka je určena na základě dvou proměnných MPS (more probable symbol) a LPS (less probable symbol). [11]

Pro kompresi používá JBIG2 dokument, který může obsahovat jednu i více stránek. V Typické stránce jsou různé typy dat - text, pultónové obrazy a ostatní (generická) data. [11]

JBIG2 model zachází s textovými daty a půltónovými obrazy jako se speciálními případy. Proto se předpokládá, že JBIG2 enkodér rozdělí obsah stránky na oblast textovou, oblast půltónovou a generickou. Některé oblasti mohou být prázdné a různé oblasti se mohou na fyzické stránce i překrývat. [11]

Kódování textové oblasti je založeno na zakódování bitmapy jednoho reprezentativního symbolu, namísto kódování každého výskytu jednotlivě. Bitmapové reprezentace symbolů se ukládají do jednoho či více slovníku. Slovník obsahuje indexovaný seznam bitmap, kde každá bitmapa reprezentuje právě jeden symbol. Při použití bezztrátové komprese se berou symboly i s nepatrným rozdílem v jejich bitmapové reprezentaci jako různé. Naproti tomu u ztrátové komprese se odchylky neberou v úvahu. Velikost odchylek závisí na nastavení míry ztrátové komprese. [11]

Pro kompresi textově zobrazených dat se používají dvě metody: [11]

- metoda shody se vzorem a substituce
- metoda jemné shody se vzorem.

U metody shody se vzorem a substituce mohou nastat dvě situace. Když je nalezena shoda s informací ve slovníku, pak se vezme index reprezentující bitmapu pro daný symbol (zpravidla první nalezená bitmapa pro symbol) a ten se zakóduje spolu s pozicí symbolu na stránce. Pokud není nalezena shoda, pak se zakóduje přímo do slovníku. Ačkoliv tato metoda dosahuje velmi dobrého kompresního poměru, mohou nastat substituční chyby při nízkém rozlišení kódovaného zobrazení. [11]

U metody jemné shody se vzorem jsou navíc potřeba upravená data (očištěná), protože mají vysoký význam při rekonstrukci originálního symbolu. Upravená data obsahují požadovaný symbol, který je zakódován pomocí kódovaného symbolu a symbolu ve slovníku. Díky tomuto je vznik substituční chyby vysoce nepravděpodobný. [11]

Půltónová oblast se skládá z množství obrazců umístěných podél pravidelné mřížky. Obrazce obvykle odpovídají hodnotám odstínu šedi. Komprese může být realizována dvěma metodami. Jedna z metod je vysoce podobná kontextově závislému aritmetickému kódování, který přizpůsobuje určení pozice jednotlivých vzorů za účelem získání vztahu mezi přiléhajícími pixely. U druhé metody se půltónové zobrazení převede zpět do odstínu

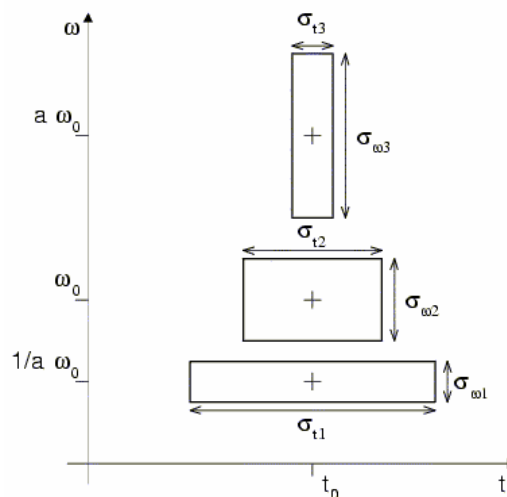
šedi. Hodnoty převedených odstínu šedi jsou pak použity jako indexy slovníku s pultónovými obrazci (malé bitmapové obrazce pevné velikosti). [11]

Pro generickou oblast se používá kontextově závislé aritmetické kódování. [11]

### 3.10 Vlnková transformace

Vznik vlnkové transformace (wavelet transform) je jedním z výsledků snahy získat časově-frekvenční popis signálu [13]. Vlnková transformace se využívá např. při kompresích obrazů nebo při zvýrazňování signálů v šumu. Existuje několik způsobů výpočtu vlnkové transformace s různou výpočetní náročností pro různé typy použitých vlněk. Mezi možné způsoby patří výpočet pomocí konvoluce, výpočet pomocí FFT nebo pomocí algoritmu zvaný lifting. [14]

Historicky starší Fourierova transformace poskytuje informaci o tom, které frekvence se v signálu nacházejí, nevypovídá však o jejich umístění (poloze) v čase, je tedy vhodná jen pro popis stacionárních signálů. [13]



Obr.20 Časově-kmitočtové rozlišení vlnkové transformace [13]

Možným řešením uvedeného problému je použití okna, které v čase ohraničí krátký úsek signálu a umožní z něj určovat spektrum v daném časovém intervalu (tento postup se nazývá Short-Time Fourier Transform, varianta s Gausovským oknem Gaborova

transformace, Gabor Transform). Z obdoby Heisenbergova principu neurčitosti vyplývá, že nelze současně určit přesně frekvenci a polohu jejího výskytu v čase. Proto má uvedené řešení pro časově konstantně široké okno pro všechny kmitočty velkou rozlišitelnost ve frekvenci a malou v čase a naopak pro časově úzké okno velkou rozlišitelnost v čase a malou ve frekvenci. Ideou vlnkové transformace je vhodnou změnou šířky okna v čase a jeho tvarem dosáhnout optimálního poměru rozlišitelnosti v čase a frekvenci. Pro nízké frekvence je okno širší, pro vysoké užší - viz *Obr.20*. [13]

Okno se nazývá mateřská vlnka  $\psi$  (mother wavelet), výpočet prototypu vlnky se provede následovně: [13]

$$\psi_{\tau,s}(t) = \frac{1}{\sqrt{s}} \psi\left(\frac{t-\tau}{s}\right) \quad s, \tau \in \mathbb{R}; s \neq 0$$

Pomocí parametru  $s$ , který se jmenuje měřítko, je možné měnit její šířku (dilatace), parametrem  $\tau$  zvaným poloha, se mění umístění vlnky na časové ose (translace). Člen  $1/\sqrt{s}$  slouží k normalizaci energie vlnky při změnách měřítka,  $\psi(t)$  je tzv. prototyp vlnky. [13]

Spojité vlnkové transformace (Continuous Wavelet Transform - CWT) je pak definována pro signály s konečnou energií, tzn.  $f \in L^2(\mathbb{R})$ , takto: [13]

$$Wf(\tau, s) = \int_{-\infty}^{\infty} f(t) \overline{\psi_{\tau,s}(t)} dt$$

kde  $\overline{\psi}$  označuje číslo komplexně sdružené.

Výsledkem pro jednorozměrný signál je dvojrozměrná funkce, která se nazývá vlnkové koeficienty  $Wf(\tau, s)$ . Po dosazení získáme tvar: [13]

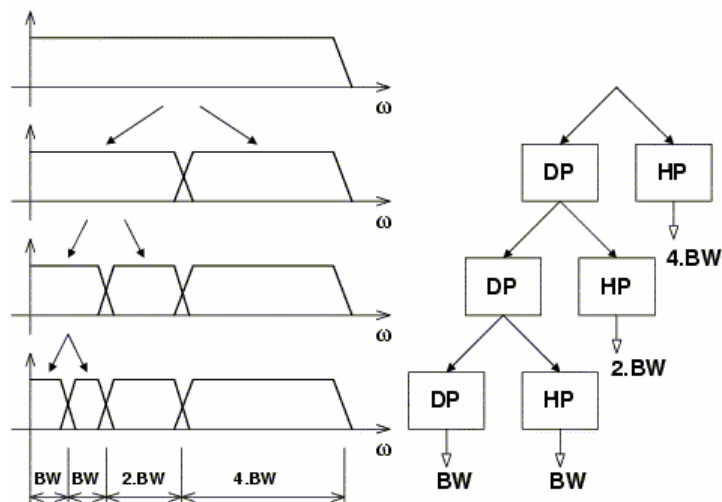
$$Wf(\tau, s) = \int_{-\infty}^{\infty} f(t) \frac{1}{\sqrt{s}} \overline{\psi\left(\frac{t-\tau}{s}\right)} dt$$

### 3.10.1 Diskrétní vlnková transformace

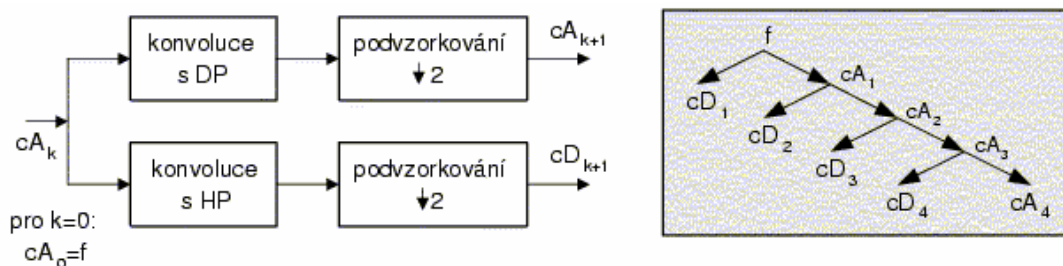
Vhodnou dvojkovou závislostí parametrů  $s$  a  $\tau$  můžeme vytvořit z vhodné vlnky  $\psi$  ortonormální bázi. Když  $s=2^p$ ;  $\tau=2^p k$ ;  $p, k \in \mathbb{Z}$  ( $p$  - měřítko;  $k$  - poloha), tak: [13]

$$\psi_{k,p}(t) = \frac{1}{\sqrt{2^p}} \psi\left(\frac{t-2^p k}{2^p}\right)$$

Díky ortonormalitě pak takto zvolená vlnka umožňuje neredundantní dekompozici signálu, tzv. analýzu s mnoha rozlišeními (multiresolution analysis, decomposition). Tento princip je základem diskrétní vlnkové transformace (Discrete Wavelet Transform - DWT). Vlnková funkce  $\psi$  se chová jako pásmová propust filtrující vstupní signál kolem centrálního kmitočtu, který je závislý na měřítku mocninou dvou. V následujícím měřítku je filtrována horní polovina pásma předchozí dolnofrekvenční části signálu - viz *Obr.21*. S rostoucím kmitočtem roste šířka pásma (BW) tohoto filtru, činitel jakosti  $Q$  je tak konstantní pro celou množinu měřítkem odvozených filtrů. Pro zvolené minimální měřítko však zůstává nepokryto pásmo od nižších kmitočtů do nuly. Proto je od vlnky  $\psi$  odvozena měřítková funkce  $\phi$  (scaling function), která má charakter dolní propusti. [13]



*Obr.21 Frekvenční pohled na diskrétní vlnkovou transformaci [13]*



*Obr.22 Jeden krok DWT a rozklad na aproximace a detaily [13]*

DWT lze počítat rychlým algoritmem, tvořeným filtrací FIR filtry a podvzorkováním (decimací). Jeden krok DWT je vidět na *Obr.22-vlevo* a rozklad na aproximace a detaily na *Obr.22-vpravo*. [13]

Oba filtry, dolní propust (scaling filter) a horní propust (wavelet filter), tvoří pár kvadraturních zrcadlových filtrů (QMF), které mají komplementární propustná pásma. Výstupy obou filtrů jsou podvzorkovány na polovinu vstupních vzorků. Horní propust poskytuje koeficienty tzv. detailů DWT (cD), dolní propust koeficienty tzv. aproximace (cA). Díky decimaci je celkový počet koeficientů po jednom kroku stejný jako počet vstupních vzorků. Koeficienty aproximace lze dále analyzovat shodným rozkladem filtry a obdržet tak další soubor koeficientů aproximace a detailů. Tak lze postupovat až do vyčerpání vstupní sekvence. [13]

Protože je počet koeficientů shodný s počtem vzorků a nedochází ke ztrátě informace, popis signálu je neredundantní. Popis je také úplný, pomocí inverzního postupu k postupu na *Obr.22* lze přesně rekonstruovat analyzovaný signál. Inverzní diskrétní vlnková transformace se označuje IDWT. Operace podvzorkování je nahrazena převzorkováním, kdy za každým vzorkem původní sekvence následuje doplněný nulový vzorek. Místo původních filtrů jsou použity rekonstrukční filtry. Výsledná aproximace cAp je použita spolu se vstupními detaily cDp jako vstup dalšího kroku IDWT (jde o pohyb nahoru ve schématu pyramidálního rozkladu v *Obr.22-vpravo*). [13]

### 3.10.2 Celočíslná vlnková transformace

Celočíslná vlnková transformace (Integer Wavelet Transform - IWT) je reverzibilní, tzn. obraz může být z celočíselných koeficientů plně rekonstruován. Lze ji tedy použít pro neztrátovou kompresi, ale kvantizováním koeficientů je možné provést i kompresi ztrátovou. [8]

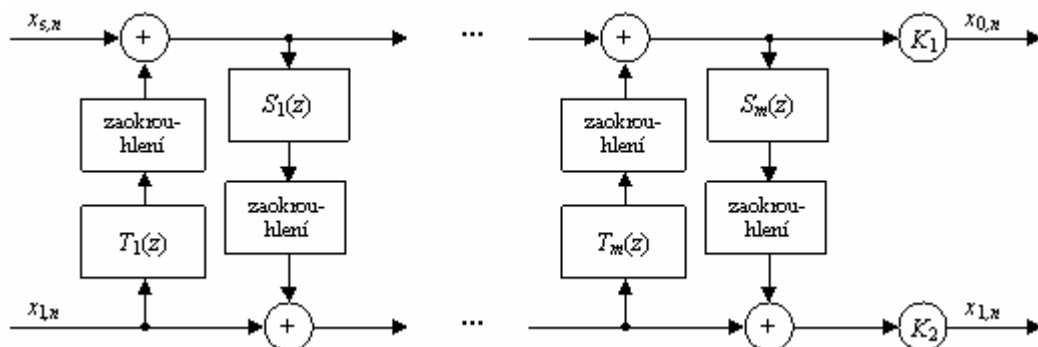
Celočíslná vlnková transformace může být použita k rozložení obrazu kterýmkoliv způsobem jako vlnková transformace s reálnými čísly - např. pyramidovým rozkladem, rozkladem na linie atd. [8]

Je několik způsobů, jak modifikovat DWT tak, že produkuje celočíselné koeficienty - jedním z nich je např. modifikovaný algoritmus lifting [8]. Při výpočtu vlnkové transformace pomocí algoritmu lifting je možné tento algoritmus upravit tak, aby jeho

výstupní hodnoty byly celočíselné. Pokud se upraví i rekonstrukční část vlnkové transformace je výsledná transformace reverzibilní. Vlastní úprava spočívá v zavedení zaokrouhlování do jednotlivých stupňů liftingu. [14] [15]

Algoritmus lifting je založen na vyjádření rozkladových a rekonstrukčních filtrů pomocí jejich polyfázových ekvivalentů. Dále jsou tyto polyfázové ekvivalenty rozkládány na jednotlivé kroky liftingu. Rozklad na jednotlivé kroky nemusí být jednoznačný a může být možný několika způsoby. V případě využití tohoto algoritmu při vlnkové transformaci s dlouhými vlnkami, je jeho výpočetní náročnost poloviční oproti standardní metodě výpočtu pomocí konvoluce. [14] [15]

Blokové schéma celočíselného vlnkového rozkladu je na *Obr.23*. Přenosové funkce  $S(z)$  a  $T(z)$  mohou být libovolné a nezáleží ani na počtu kroků liftingu. V blokovém schématu se vyskytuje násobení konstantami (blok  $K_1$  a  $K_2$ ), které mohou způsobit vznik čísel s desetinou čárkou, pokud jejich hodnota není celé číslo. V takovém případě se celočíselné posloupnosti na výstupu získají vynecháním těchto násobiček, ale musejí se vynechat i násobičky v rekonstrukční části. Pokud se vynechají, tak ovšem výstupní signály neodpovídají přesně jejich hodnotám, ale jistým násobkům a musí se k tomu upravit algoritmus zpracovávání dat po vlnkové transformaci. [14] [15]

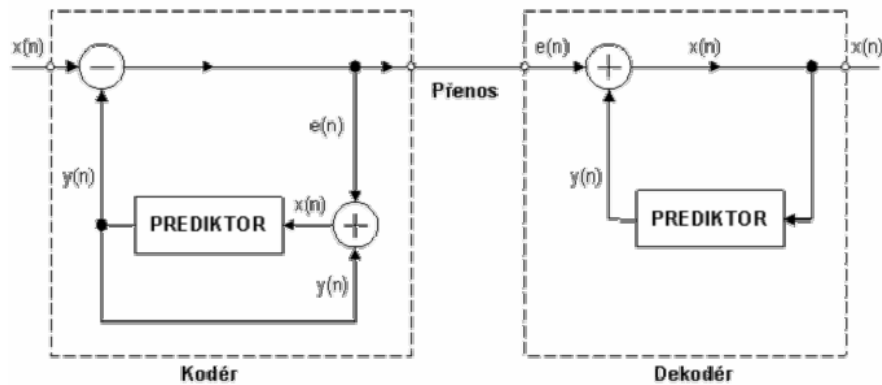


*Obr.23* Blokové schéma několikastupňového celočíselného liftingu [15]

### 3.11 Prediktivní metody

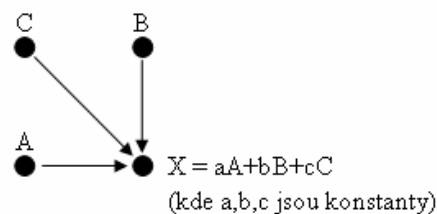
Hodnoty sousedních bodů digitálního obrazu jsou si velmi často podobné (existuje mezi nimi tzv. prostorová korelace). Díky této závislosti lze odhadnout (predikovat) hodnotu

následujícího bodu v obraze z hodnot sousedních bodů. Při úspěšné predikci se odhadovaná hodnota  $y(n)$  jen málo liší od skutečné  $x(n)$ . Tento rozdíl je takzvaná chyba predikce  $e(n)$ . Tato chyba dosahuje menších absolutních hodnot než skutečná hodnota pixelu a k jejímu zakódování tedy postačuje kratší kódové slovo. Pro pohyblivý obraz lze predikovat aktuální hodnotu pixelu z hodnot téhož pixelu v předchozích snímcích (důsledek tzv. časové korelace). Blokové schéma predikčního kodéru využívajícího principů rozdílové pulsně kódové modulace (DPCM - Differential Pulse Code Modulation) je na *Obr.24*. [16]



*Obr.24* Blokové schéma predikčního kodéru a dekodéru [16]

Predikce současné hodnoty může být určena např. váhovanou lineární kombinací předchozích vzorků nebo pomocí polynomiálních funkcí či dvojrozměrných prediktorů. Princip dvojrozměrného prediktoru je zobrazen na *Obr.25*. [16]



*Obr.25* Princip dvourozměrné predikce [16]

### 3.11.1 PPM

PPM (Prediction by Partial Matching) je adaptivní statistická technika datové komprese založená na kontextovém modelování a predikci. PPM modely používají množinu předchozích symbolů pro predikci následujícího symbolu. [17]

Cestou, jak při kódování přesněji určit frekvenci výskytu znaku, je zjištění kontextu, ve kterém se znak vyskytuje - proto se metoda PPM také nazývá metoda konečného kontextu. Např. při kompresi běžného textu v českém jazyce se písmeno ‚e‘ vyskytuje v mnohem větší míře po písmenu ‚s‘ než třeba po písmenu ‚w‘. Kontextová metoda pro kódování výskytu ‚e‘ po ‚s‘ použije menší počet bitů než pro kódování výskytu ‚e‘ po ‚w‘. Vzhledem k tomu, že tyto metody používají adaptivní model, kde kontext musí mít k dispozici i dekompresní algoritmus, připadá v úvahu jen levý kontext kódovaného znaku. [6]

Problémem kontextového kódování je celkový počet frekvencí, který velmi narůstá s délkou kontextu. U bezkontextového kódování máme 256 různých znaků a tím i 256 frekvencí. Pokud bychom vzali jako kontext jeden znak, máme již  $256^2$  možných frekvencí, neboť každý z 256 znaků může mít v textu před sebou kterýkoliv z 256 znaků. Protože zvětšení kontextu o 1 způsobí zvýšení možných frekvencí o mocninu 256, tak kontextové metody používají jen omezenou délku kontextu - typicky nebývá délka kontextu větší než 4. [6]

Dalším problémem kódování v kontextu je situace, kdy se znak v daném kontextu vyskytuje poprvé. Frekvence jeho dosavadního výskytu a tedy i pravděpodobnost je v tomto případě nula. Nulovou pravděpodobnost ale statisticky kódovat nelze. Proto je nutné pro tyto případy vyčlenit nějakou nenulovou hodnotu pravděpodobnosti. Jak velkou tuto pravděpodobnost zvolit je určité dilema, neboť tato má nezanedbatelný vliv na účinnost komprese. Protože není k dispozici způsob, jak výpočet této pravděpodobnosti matematicky odvodit, bylo intuitivně navrženo několik možných přístupů řešení uvedeného problému. Experimentálně bylo ověřeno, že nejlepší výsledky dává metoda označená jako C. [6]

Níže popsaná metoda konečného kontextu je známa pod označením PPMC. Poslední písmeno označuje, že pro výpočet escape kódu se používá metoda C. [6]

Pravděpodobnost nového znaku  $z$  (znaku, který se v daném kontextu  $\alpha$  doposud v textu nevyskytl) metoda C stanoví dle vztahu: [6]

$$e_k(\alpha) = \frac{c_k(\alpha)}{f_k(\alpha) + c_k(\alpha)}$$

$c_k(\alpha)$  - počet všech různých znaků, které se doposud vyskytly v kontextu  $\alpha$ .

$f_k(\alpha)$  - počet výskytů (frekvence) kontextu  $\alpha$  s délkou  $k$ .

$f_k(\alpha)$  je součet frekvencí  $f_k(\alpha, z)$

$f_k(\alpha)$  se vypočte vztahem: [6]

$$f_k(\alpha) = \sum_z f_k(\alpha, z)$$

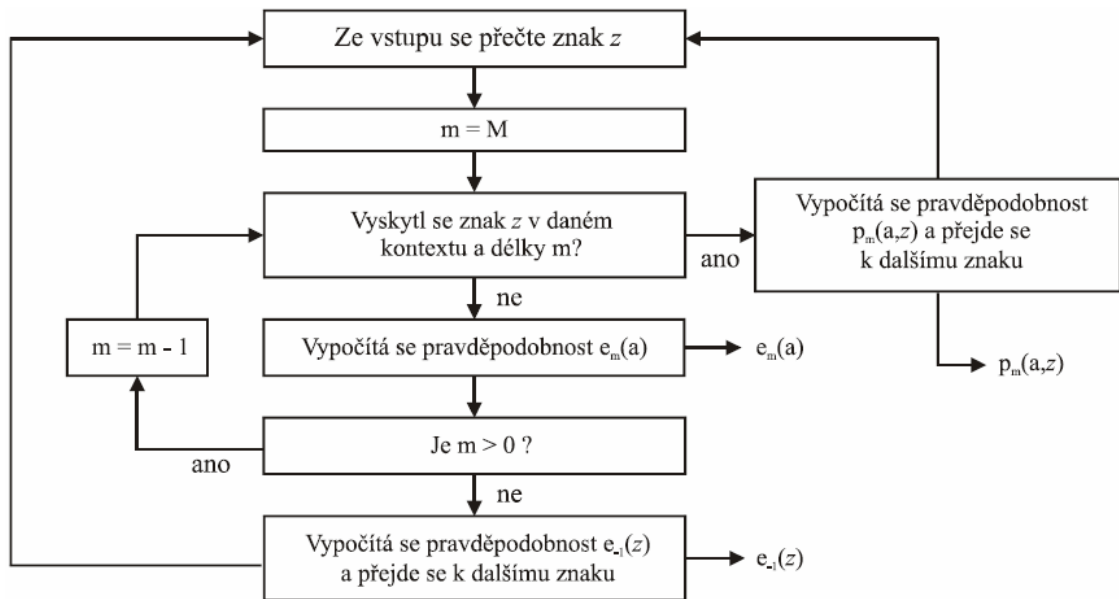
$f_k(\alpha, z)$  - počet výskytů (frekvence) znaku  $z$  v kontextu  $\alpha$  o délce  $k$ , přičemž  $z$  v sumě označuje všechny znaky, které se v kontextu  $\alpha$  již vyskytly, tj. u kterých je  $f_k(\alpha, z) > 0$ . [6]

Při stanovení nenulové pravděpodobnosti  $e_k(\alpha)$  se sníží prostor pro pravděpodobnosti znaků, které se již v kontextu  $\alpha$  vyskytly. Výpočet pravděpodobností těchto znaků se musí upravit tak, aby součet všech pravděpodobností v kontextu  $\alpha$  zůstal roven 1. Metoda C stanoví pro znaky, které se již v kontextu vyskytly, výpočet pravděpodobností výskytu znaku  $z$  v kontextu  $\alpha$  o délce  $k$  dle vztahu: [6]

$$p_k(\alpha, z) = \frac{f_k(\alpha, z)}{f_k(\alpha) + c_k(\alpha)}$$

Z uvedených vztahů plyne, že čím více se nějakých znaků před novým znakem v daném kontextu již vyskytlo (hodnota  $f_k(\alpha)$ ), tím menší bude pravděpodobnost  $e_k(\alpha)$ . Intuitivně to vyjadřuje skutečnost, že pokud se znak doposud v nějaké delší části textu v daném kontextu zatím nevyskytl, bude jeho pravděpodobnost výskytu v tomto kontextu celkově zřejmě malá. Naopak ale čím více těchto znaků v tomto kontextu bylo různých (hodnota  $c_k(\alpha)$ ), tím bude zřejmě hodnota pravděpodobnosti výskytu ještě dalšího nového znaku větší. [6]

Kódování probíhá podle schématu na Obr.26. Maximální délka sledovaného kontextu je označena písmenem  $M$ . Písmeno  $m$  v něm označuje aktuální délku kontextu. [6]



Obr.26 Postup komprese PPMC [6]

Kódování každého znaku začíná vždy od nejdelšího kontextu  $m = M$ . Jestliže se znak v něm již vyskytl, vypočítá se pravděpodobnost  $p_m(a,z)$ , zašle se na výstup a přechází se na další znak. Pokud ne, vypočítá se pravděpodobnost  $e_m(a)$ , zašle se na výstup a přejde se na kontext o 1 nižší. Tento proces se cyklicky opakuje do té doby, než se najde taková délka daného kontextu, ve které se znak  $z$  již vyskytl, nebo se zjistí, že znak  $z$  se v textu ještě vůbec nevyskytl a pak se na závěr zakóduje hodnota  $e_{-1}(a)$ . Výstupem metody konečného kontextu jsou tedy pravděpodobnosti výskytu jednotlivých znaků, které jsou následně zakódovány použitím aritmetického kódování. [6]

Dekompresní algoritmus si vytváří stejný model jako algoritmus kompresní. Po dekódování každého znaku dekompresní algoritmus model rovněž stejným způsobem aktualizuje. Proto se čas dekomprese příliš neliší od času potřebného pro kompresi. [6]

Metoda PPMC patří mezi nejúčinnější kompresní techniky. Pro kompresi i dekompresi však potřebuje značné množství paměti a je časově velmi náročná. [6]

### 3.12 Burrows-Wheelerova transformace

Metoda Burrows-Wheelerovy transformace (BWT) je také známa pod názvem metoda komprese blokovým tříděním. Tato metoda je založena na transformaci bloku textu o  $n$

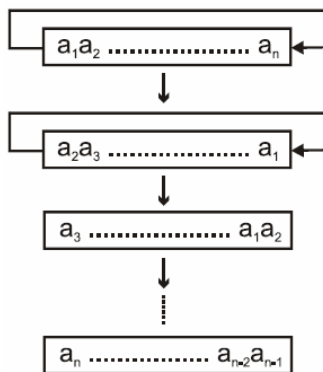
znacích  $a_1a_2\dots a_n$  na blok stejné délky vytvořený záměnou pořadí (permutací) znaků původního bloku  $a_1a_2\dots a_n$ . Transformace je volena tak, že splňuje vlastnosti: [6]

- je reverzibilní, tzn. text se dá zpětně převést do původní podoby.
- po transformaci se stejné znaky velmi často vyskytují za sebou, což umožňuje jejich efektivní kódování.

Metoda blokového třídění je založena na transformaci bloku textu do podoby, ve které se často objevují sekvence stejných znaků za sebou. Proto účinnost metody roste s délkou textu. Ideální je provést transformaci a kódování celého textu najednou. Značně dlouhé texty ale bude nutné rozdělit do bloků a jednotlivé bloky zpracovat samostatně. Délka bloku se volí dostatečně dlouhá - řádově statisíce znaků. [6]

Jádrem metody je setřídění cyklických rotací textu v bloku. Vzhledem k tomu, že blok textu bývá značně dlouhý, byla by vlastní matice rotací bloku neúnosně rozsáhlá. Proto se místo ní používá sloupcový vektor s čísly, které vyjadřují kolik znaků rotace v každém řádku původní matice obsahuje. Řádek s nerotovaným textem je reprezentován číslem 0, řádek s jedním rotovaným znakem číslem 1 atd. [6]

Při kompresi se text (resp. hodnoty textu) v komprimovaném bloku  $a_1a_2\dots a_n$  cyklicky posune doleva, čímž se vytvoří cyklická rotace bloku textu. Pokud se bude pokračovat v cyklických posunech, dojde dohromady k  $n-1$  různých rotací bloku textu - spolu s výchozím blokem je to tedy celkem  $n$  bloků textu. Znázornění této rotace je vidět na *Obr.27*. [6]



*Obr.27* Vytvoření rotace bloku textu [6]

Setříděné rotace bloku o  $n$  znacích tvoří čtvercovou matici znaků  $M$  řádu  $n$ . Každý sloupec matice obsahuje právě všechny znaky komprimovaného bloku textu, ale v jiném pořadí. Jako požadovaná transformace se vezme poslední sloupec této matice a dále číslo řádku, na kterém je původní text. [6]

V další fázi se vezmou všechny znaky, které se mohou vyskytnout v textu, a uspořádají se abecedně do posloupnosti  $Z$ . Následně se postupně zleva z transformovaného bloku odebírají jednotlivé znaky. U každého znaku se zjistí jeho pozice v posloupnosti  $Z$  a číslo této pozice se zakóduje na výstup. Po zakódování každého znaku se provede cyklická rotace posloupnosti  $Z$  tak, aby se právě zakódovaný znak dostal na počáteční pozici. [6]

Výstupem komprese je: [6]

- Číslo řádku s původním textem
- Hodnoty výstupního kódu  $K$

Při dekompresi se z komprimovaného souboru přečte číslo řádku, na kterém je v matici  $M$  původní text, a pomocí aritmetického dekodéru se dekodují hodnoty uloženého kódu  $K$ . [6]

Dále se vezme posloupnost znaků abecedy  $Z$  a z kódu  $K$  se získá výchozí transformovaný blok textu, což je poslední sloupec matice znaků  $M$ . Pomocí hodnoty kódu  $K$  se najde příslušný znak v posloupnosti  $Z$  a cyklickou rotací se tento znak v posloupnosti přesune na její začátek stejným způsobem, jak se to dělalo při kódování. [6]

Posledním krokem je zpětná transformace bloku textu, jejímž výsledkem bude původní text. Je znám poslední sloupec matice znaků  $M$ , první sloupec matice  $M$  obsahuje stejné znaky jako poslední sloupec a znaky jsou v něm abecedně setříděné. Tudíž setříděním posledního sloupce podle abecedy se získá první sloupec matice  $M$ . Nyní se sestaví funkce  $T$ , která mapuje řádky posledního sloupce na řádky prvního sloupce s vlastnostmi: [6]

- Řádek posledního sloupce se vždy zobrazí na řádek prvního sloupce se stejným znakem
- Na řádcích posledního sloupce, na kterých je stejný znak, je funkce  $T$  rostoucí. Tzn. pro indexy řádků  $i, j$ , pro něž jsou znaky na řádcích  $i$  a  $j$  stejné, platí, že jejich funkční hodnoty jsou  $T(i) < T(j)$  právě když je  $i < j$ .
- Funkce je prostá (bijektivní)

## 4 FORMÁTY VYUŽÍVAJÍCÍ NEZTRÁTOVOU KOMPRESI

Existence mnoha formátů má několik příčin: [1]

- Historické důvody - formáty odrážejí technický vývoj, zejména postupně se zvyšující barevné možnosti zařízení jak pro snímání obrazu, tak pro jeho zobrazení.
- Vazba na program - podle druhu aplikace vznikaly specializované formáty, například pro úschovu skic a kreseb (PCX), černobílých dokumentů (TIFF) či pro přenos barevných fotografií a jejich prezentaci na WWW (GIF).
- Technické důvody - mnoho formátů bere ohled na rozlišení skenerů, pomocí kterých jsou obrazy zaznamenávány, na obrazová rozlišení v různých osách, na odlišné architektury obrazové paměti v grafických kartách apod.
- Metoda komprese - vzhledem k velkému paměťovému objemu barevných obrazů je žádoucí uchovávat obraz v komprimované podobě. Volba vhodné kompresní metody je často závislá na charakteru obrazu a na jeho dalším použití.

Uvedený přehled nezaznamenává zcela všechny důvody existence široké škály obrazových formátů. Existuje celá řada dalších aplikačních požadavků, jakými je například uložení více obrázků v jednom souboru, zápis sekvence obrazů pro animaci apod. [1]

### 4.1 BMP

BMP je velmi rozšířený formát, který nejčastěji obsahuje nekomprimovaná data. Je zde však možné použít komprimaci RLE. Komprimaci lze využít pouze u bitmap se čtyřmi a osmi bity na pixel. U 1 a 24 bitových bitmap jsou obrázky uloženy v nekomprimované podobě. Formát pracuje s uspořádáním Bytů malý endian. [18]

Formát BMP umožňuje ukládání rastrových dat ve čtyřech bitových hloubkách: [18]

- 1 bit na pixel s barevnou paletou o délce 8 Bytů
- 4 bity na pixel s barevnou paletou o délce 64 bytů
- 8 bitů na pixel s barevnou paletou o délce 1024 Bytů
- 24 bitů na pixel s reprezentací pixelu přímo barvou BGR.

Každý korektní soubor typu BMP obsahuje sekce popsané v *Tab.2*, ve které jsou sekce uvedeny podle pořadí výskytu v souboru. Barevnou paletu neobsahují obrázky bitové hloubky 24 bitů. [18]

*Tab.2 Struktura formátu BMP [18]*

Název	Význam
BITMAPFILEHEADER	hlavička souboru BMP
BITMAPINFOHEADER	informační hlavička o obrázku
RGBQUAD[]	tabulka barev (barevná paleta)
BITS	pole bitů obsahujících vlastní rastrová data (pixely)

Hlavička souboru formátu BMP (BITMAPFILEHEADER) má délku 14 Bytů. Obsah hlavičky je vidět v *Tab.3*. [18]

*Tab.3 Hlavička formátu BMP [18]*

Offset	Velikost [Byte]	Význam
0	2	identifikátor - vždy obsahuje čísla 42h 4Dh
2	4	celková velikost souboru. Některé aplikace tuto položku ignorují a dosazují zde 0.
6	2	rezervovaný pro pozdější použití - vždy 0
8	2	rezervovaný pro pozdější použití - vždy 0
10	4	offset začátku obrazových dat

Hlavička s informacemi o obrázku (BITMAPINFOHEADER) obsahuje základní metainformace o rastrovém obraze. Tato hlavička je popisána *Tab.4* [18]. Hlavička verze 3 je nejčastěji používána, i když je možné se setkat i s novějšími hlavičkami verze 4 a 5 [19].

Tab.4 Informační hlavička formátu BMP verze 3 [18]

Offset	Velikost [Byte]	Význam
0	4	celková velikost informační hlavičky - vždy 40
4	4	šířka obrazu v pixelech
8	4	výška obrazu v pixelech
12	2	počet bitových rovin - vždy 1
14	2	počet bitů na pixel - 1, 4, 8 nebo 24
16	4	typ komprimační metody (0 - bez komprese, 1 - RLE8, 2 - RLE4)
20	4	velikost obrazu v Bytech - pokud je bitmapa nekomprimovaná, může zde být 0
24	4	horizontální rozlišení výstupního zařízení v pixelech na metr
28	4	vertikální rozlišení výstupního zařízení v pixelech na metr
32	4	celkový počet barev, které jsou použité v dané bitmapě. 0 znamená použití maximálního počtu barev
36	4	počet barev, které jsou důležité pro vykreslení bitmapy. 0 znamená, že jsou všechny barvy důležité

V barevné paletě udává první trojice Bytů hodnotu tří barevných složek posloupnosti BGR a čtvrtý Byte je nastavený na nulovou hodnotu. Čtvrtý Byte není možné (podle platné specifikace) použít pro jiné účely - tedy ani pro alfa kanál. Podpora alfa kanálu je zavedena až od Windows XP, ale mnoho programů ji neimplementuje. [18]

Ukládání obrazových řádků do souborů se provádí zleva doprava a zespodu nahoru. Všechny obrazové řádky musí mít Bytovou velikost dělitelnou čtyřmi. Mohou zde být uloženy indexy odkazující na paletu, pixely ve formátu BGR a nebo komprimovaná data. [18]

U RLE jsou obrazová data organizována do skupiny po dvou Bytech. První Byte určuje počet po sobě jdoucích pixelů, které budou mít index barvy určený v druhém Bytu. První Byte ve skupině může být nastavený na nulu, což znamená tzv. únik: například konec řádku, konec bitmapy nebo delta. Typ úniku je zapsán ve druhém Bytu skupiny. Význam jednotlivých únikových hodnot je v Tab.5. [19]

Tab.5 Hodnoty úniku formátu BMP [19]

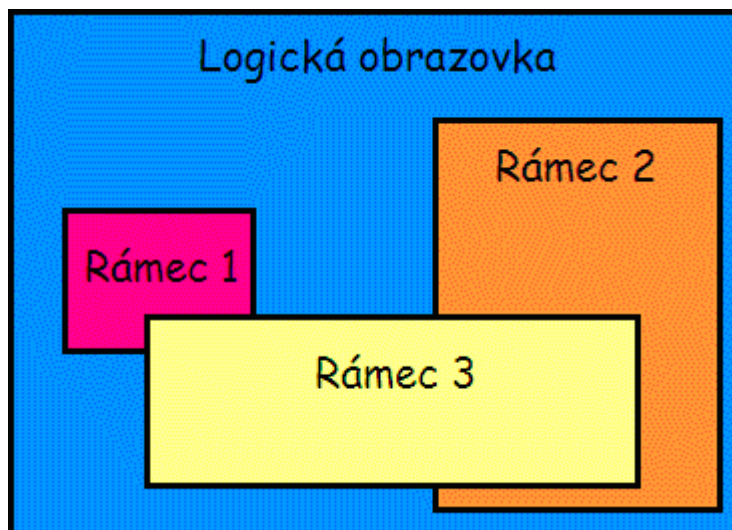
Hodnota	Význam
00h	konec řádku
01h	konec bitmapy
02h	delta - následující 2 Byty znamenají horizontální a vertikální posunutí následujících pixelů
03h - FFh	počet indexů barev, které následují v nezkomprimovaném tvaru

## 4.2 GIF

Grafický formát GIF byl vyvinut firmou CompuServe. Tento formát využívá LZW kompresi. [3] Zajímavostí je, že komprese LZW byla zatížena dvěma patenty: firmy Unisys a IBM. V USA vypršel patent Unisysu dne 19.6.2003 přesně v 0:00. V dalších zemích, například v Evropě (Německo, Velká Británie, Francie, Itálie) či Japonsku patent vypršel o rok později, tj. o půlnoci 19.6.2004. Poslední známý patent (ve vlastnictví IBM) vztahující se ke formátu GIF vypršel dne 11.8.2006. Od daného dne je tedy možné grafický formát GIF bez omezení používat v programových aplikacích. [20]

Maximální rozlišení obrázku formátu GIF by mělo být 65535×65535 pixelů, protože pro uložení informace o šířce a výšce jsou využity pro každý údaj 2 Byty [22]. Tuto teorii podporuje také zdroj [21], který dále říká, že lze pro tento údaj nalézt hodnoty 16000×16000, 2048×2048. V knize [3] str.371 je napsáno: „Maximální velikost předlohy: 42Kb x 64Kb pixelů“. Ve specifikaci formátu GIF [22] se však žádný údaj o maximálním rozlišení obrázku nevyskytuje, a tak je možné předpokládat, že je šířka i výška omezena maximální hodnotou dvou Bytů - tedy 65535.

V grafickém formátu GIF se celý obrázek, který je zde nazýván logická obrazovka, skládá z několika tzv. rámců, a není tedy v souboru uložen jako jeden celek. Rámce jsou obdélníkové oblasti umístěné uvnitř logické obrazovky, jak je také vidět na *Obr.28*. Není nutné, aby rámce pokryly celou logickou obrazovku a mohou se navzájem překrývat. [21]



Obr.28 Logická obrazovka a rámce formátu GIF [21]

Minimálně musí být vždy přítomen jeden rámeček, jejich maximální množství však není omezeno. Každý rámeček je možné chápat jako rastrový obrázek, který je celou svou plochou umístěn v logické obrazovce - podle specifikace nesmí žádný pixel z rámce padnout mimo logickou obrazovku. Pozice rámce v logické obrazovce je určena souřadnicí jeho horního levého rohu a velikostí (šířkou, výškou) zadanou v pixelech. [21]

Vzhledem k tomu, že každý pixel umístěný v rámci může být popsán maximálně osmi bity, znamená to, že jeho barvu je možné vybrat z barevné palety obsahující nejvýše 256 barev. Každá barva je v barevné paletě popsána trojicí hodnot - barevných složek označených písmeny RGB, přičemž každá barevná složka je uložena v jednom Bytu. Minimální velikost barvové palety (se dvěma barvami) je proto rovna šesti Bytům ( $2 \times 3$ ), maximální velikost barvové palety s 256 barvami pak 768 Bytům ( $256 \times 3$ ). [21]

V grafickém formátu GIF rozlišujeme dvě barevné palety: globální a lokální. Globální barevná paleta může v souboru formátu GIF existovat maximálně jednou a její velikost (tj. počet barev) je zadána v hlavičce popisující logickou obrazovku. V některých případech nemusí být globální barevná paleta přítomna vůbec. [21]

Lokální barevná paleta může být přiřazena ke každému rámci, opět se však nejedná o povinnou součást rámce. V případě, že není přítomna ani globální ani lokální barevná paleta, měl by prohlížeč použít systémovou paletu, v případě webových prohlížečů takzvanou web-safe paletu složenou z 216 barev rovnoměrně umístěných

v jednotkové RGB krychli a zbylých 40 barev odstínů šedi. Absence obou barevných palet sice zmenší celkovou velikost souboru, způsob zobrazení se však může v různých prohlížečích lišit, proto se většinou alespoň jedna barevná paleta používá. [21]

I když se v literaturách uvádí (např. v [3]), že v obrázku formátu GIF je možné zobrazit pouze 256 různých barev, není to podle [21] pravda. Pravdivé je pouze tvrzení, že v jednom rámci může být zobrazeno maximálně 256 barev. Pro zvýšení celkového počtu barev lze využít možnosti lokálních barevných palet. [21]

Důležitou součástí popisu logické obrazovky je index jedné barvy v globální barevné paletě, která bude použita pro vykreslení pozadí. Dodnes sice většina programů při ukládání souboru formátu GIF každý obrázek uloží do jednoho velkého rámce, ale při optimalizaci na velikost je možné (a některé programy to provádí), aby některé pixely logické obrazovky nebyly pokryty žádným z rámců. Nepokryté pixely by poté měly být vybarveny právě barvou pozadí. Tímto způsobem je možné vytvořit optimalizované obrázky, které budou zabírat malé místo na disku, a to díky tomu, že je uložen např. pouze malý rámeček uprostřed velké volné plochy. [21]

Verze GIF89a přinesla oproti původní verzi GIF87a jednu podstatnou novinku - pomocí rozšiřujícího řídicího bloku (graphics control extension) je možné v každém rámci specifikovat jeden index do barevné palety, který představuje průhledné pixely. Vzhledem k tomu, že je možné průhlednými pixely překreslit i původně neprůhlednou barvu pozadí, může mít i obrázek jako celek neobdélníkový tvar. [21]

Často se říká, že GIF podporuje pouze jednobitovou průhlednost. V podstatě to je pravda, protože každý pixel v obrázku může být buď zcela průhledný nebo naopak zcela neprůhledný. Ve formátu GIF však není každému pixelu přiřazen speciální bit určující průhlednost, protože u zcela průhledných pixelů není zapotřebí specifikovat barvu. [21]

Ve formátu GIF byla zavedena možnost ukládání obrazových řádků v jednotlivých rámcích neboli tzv. režim prokládání (interlace mode). Obrazové řádky tedy nejsou v rámci uloženy ve své přirozené sekvenci (1, 2, 3 ... n), ale hierarchicky podle schématu, při jehož použití se obrázky vykreslují ve čtyřech průchodech. [21]

Jak ukazuje *Obr.29* nejprve jsou v souboru uloženy řádky 0, 8, 16, 24 atd. - tj. osmina celého rámce. Následuje druhý průchod, ve kterém jsou uloženy řádky 4, 12, 20, 28 atd. - tj. osmina rámce. Ve třetím průchodu je uložena čtvrtina řádků, a to řádky 2, 6, 10, 14 atd.

Zbývající polovina obrazových řádků je uložena ve čtvrtém a současně i posledním průchodu. [21]

1. průchod
4. průchod
3. průchod
4. průchod
2. průchod
4. průchod
3. průchod
4. průchod
1. průchod
4. průchod
3. průchod
4. průchod
2. průchod

*Obr.29 Prokládaný režim formátu GIF*

Díky prokládání může uživatel při přenosu obrázků ze serveru ke klientovi získat představu o celém obrázku již po přenesení jedné osminy dat, tj. po prvním průchodu a přenos popř. zastavit. Nevýhodou prokládání je poněkud větší délka celého souboru, neboť se sníží vzájemná podobnost pixelů mezi jednotlivými řádky, což zapříčiní horší účinnost algoritmu LZW. [21]

Celý soubor je rozdělen do bloků, z nichž některé jsou povinné, některé se musí vyskytovat na určitém místě souboru, další se mohou opakovat apod. Povolené bloky, které se mohou ve formátu GIF vyskytovat, jsou popsány v *Tab.6* spolu s informací, zda se jedná o bloky povinné (tj. bloky, které se musí vyskytovat v každém validním souboru) a zda je možné bloky v rámci jednoho souboru opakovat (pro účely animací, slideshow, obejití maximálního počtu 256 barev apod.). Dále je u každého bloku uvedeno, zda se jeho definice objevila už ve specifikaci GIF87a nebo až v novější specifikaci GIF89a. [23]

Tab.6 Bloky formátu GIF [23]

Název bloku	Povinná položka	Lze opakovat	Verze
signatura	ano	ne	87a
verze souboru	ano	ne	87a
popis logické obrazovky	ano	ne	87a
globální barevná paleta	ne	ne	87a
rozšiřující grafický blok	ne	ano	89a
popis rámce	ano	ano	87a
lokální barevná paleta	ne	ano	87a
data rámce (pixely)	ano (pro rámeček)	ano	87a
rozšiřující informace (textové, binární)	ne	ano	89a
ukončovací znak GIF souboru	ano	ne	87a

Každý soubor typu GIF musí začínat jeho signaturou, což je sekvence Bytů s hodnotami 47h, 49h a 46h. Po převodu do ASCII můžeme tuto sekvenci číst jako řetězec „GIF“. [23]

Další trojice Bytů, která udává verzi formátu GIF. V současné době (a pravděpodobně i v budoucnosti) je možné použít pouze dvě verze, první je GIF87a (sekvence Bytů 38h 37h 61h) a druhá GIF89a (sekvence Bytů 38h 39h 61h). [23]

Dále je uložen popis logické obrazovky. Nejprve je na dvou Bytech uvedena šířka obrazovky v pixelech a následuje výška obrazovky v pixelech. U hodnot šířky a výšky jsou Byty uspořádány v pořadí nižší-vyšší (tzv. little endian). Při rozlišení 100×100 pixelů bude mít tedy šířka i výška hodnotu 64h 00h [23]

Posléze následuje bitové pole, ve kterém je uvedeno, zda je použita globální barevná paleta (sedmý bit), počet bitů na pixel-1 (bity 3–6), zda je barevná paleta setříděna (bit 2) a délka barevné palety (bity 0 a 1). [23]

Skutečná délka palety se vypočte podle vzorce: [23]

$$2^{\text{hodnota}+1}$$

Další Byte obsahuje index barvy, která je použita jako pozadí v případě, že by plocha rámců nevyplnila celý obrázek. Následuje Byte, ve kterém je uveden poměr mezi výškou a šířkou pixelu. Pokud je v tomto Byte uložena nula, není poměr specifikovaný. U verze GIF87a není tento Byte využit a podle specifikace zde musí být zapsána hodnota nula. [23]

Vztah pro výpočet poměru mezi výškou a šířkou je: [23]

$$\text{Poměr Stran} = \frac{\text{Poměr Stran Pixelů} + 15}{64}$$

Dále může obrázek obsahovat globální barevnou paletu a rozšiřující grafický blok (Graphic Control Extension - GCE). Rozšiřující grafický blok se dá využít pro obrázky využívající průhlednost. Tento blok má velikost 8 Bytů. Struktura je vidět v *Tab.7*. [23]

*Tab.7 Rozšiřující grafický blok formátu GIF [23]*

Offset	Byte	Význam Bytové sekvence
0	21h	značka začátku rozšiřujícího bloku
1	F9h	identifikace rozšiřujícího bloku
2	04h	délka rozšiřujícího bloku (vždy stejná)
3	--	bitové pole s příznaky
4	--	čas zpoždění zobrazení následujícího rámce
5	--	vyšší Byte hodnoty zpoždění (v setinách sekundy)
6	--	index barvy, která bude považována za průhlednou
7	00h	ukončení rozšiřujícího bloku

Hlavička rámce je v souboru rozpoznána značkou začátku rámce, tj. Bytem s hodnotou 2Ch. Dva Byty za značkou udávají x-ovou pozici levého okraje rámce, další dva Byty pak y-ovou pozici horního okraje rámce. Pokud rámec začíná v levém horním rohu obrázku, jsou zde uloženy nulové hodnoty (00h 00h 00h 00h). Hlavička rámce pokračuje dvěma Byty se šířkou rámce a dalšími dvěma Byty s jeho výškou. [23]

Následuje bitové pole, ve kterém je uvedeno, zda se v rámci bude používat lokální barevná paleta (sedmý bit), zda je rámec prokládán (šestý bit), zda jsou barvy v lokální barevné paletě setříděny (pátý bit) a konečně délka lokální barevné palety (bity 0-2). Pokud není lokální barevná paleta uložena, je v tomto bitovém poli (resp. Bytu) hodnota 00h. Pokud by byla lokální barvová paleta přítomna, začínala by ihned za tímto Bytem. [23]

Po hlavičce rámce již začíná sekce s daty rámce (pixely). Dekodér potřebuje při své inicializaci znát počáteční velikost kódu ukládaného do hashovací tabulky. Tato velikost odpovídá počtu bitů na pixel zvýšeného o jedničku. Např. pro maximálně dvoubarevný obrázek by tedy měla být velikost nastavená na dvojku, vzhledem k implementačním detailům LZW je však nastavena na trojku, tj. kód má velikost tři bity. V souboru je tato

velikost snížena o jedničku, tj. nalezneme zde hodnotu 02h. Následuje Byte udávající velikost bloku zakódovaného pomocí LZW. [23]

Potom jsou uložena vlastní obrazová data komprimovaná algoritmem LZW. Při uložení obrázku s rozlišením 1×1 pixelů, globální barevnou paletou s hodnotami barev FFh 80h 00h a FFh FFh FFh, přičemž jediný pixel se v původní podobě odkazuje na index palety 0 a velikost kódu je nastaven na tři bity, jsou zde uloženy hodnoty 44h 01h. Po rozpisu do binární podoby lze získat bitový vzorek 01000100 00000001, který se musí správně dekódovat. Vzhledem k tomu, že je velikost kódu nastavena na tři bity, tak se bitový vzorek rozloží do trojic: 100 000 101 --- --- - (začíná se od bitu s nejnižší vahou, hranice Bytů se překračují). Vysvětlení bitových vzorků lze vidět v *Tab.8*. Vzhledem k tomu, že třetí trojice bitů označuje konec LZW kódu, tak hodnoty dalších bitů nejsou důležité - jsou zde uloženy jako výplň. [23]

*Tab.8 Bitové vzorky konkrétního souboru formátu GIF [23]*

<b>Bitový vzorek</b>	<b>Význam</b>
000	první index do barevné palety
001	druhý index do barevné palety
100	inicializace hashovací tabulky - clear code (CC)
101	konec LZW kódu - end of LZW code (EC)

Celý blok se ukončí tzv. ukončovacím znakem, který má hodnotu 00h. Nakonec následuje ještě ukončovací hodnota formátu GIF, a to hodnota 3Bh. [23]

Poměrně neznámou a prakticky nepoužívanou funkcionalitou, kterou grafický formát GIF nabízí, je možnost uložení interaktivní slideshow (více relativně samostatných obrázků) v jednom souboru typu GIF. V rozšiřujícím grafickém řídicím bloku, který se může vyskytovat před každým rámcem, je možné nastavením jednoho bitu specifikovat, zda se má prohlížeč program v daném místě zpracování zastavit a čekat na uživatelský vstup (stisk klávesy, klik myši apod.). [24]

V případě, že je současně nastavená i doba pauzy mezi zobrazením jednotlivých rámců, bude zobrazování obrázku pokračovat až v případě, že uživatel stiskl klávesu či klikl myši nebo uplynul zadaný čas zpoždění mezi snímky. Pokud není zpoždění mezi snímky zadané, měl by prohlížeč program čekat pouze na uživatelský vstup bez provádění animace.

V dokumentaci se dokonce píše, že uživatel o možnosti pokračovat v zobrazování slideshow může být informován zvukem, tj. zasláním znaku 07h (BEL) na standardní zvukový výstup. [24]

Formát GIF podporuje od verze GIF89a také animace. Ve skutečnosti nejsou animace nic jiného, než sled po sobě jdoucích rámců (které mohou ležet na sobě), mezi jejichž zobrazením je vložena krátká či delší časová prodleva. [21]

### 4.3 PCX

Grafický formát PCX vyvinula firma ZSoft Corporation, která ho v minulosti používala ve všech verzích svých grafických editorech PC-PaintBrush. Jedná se o formát využívající jednoduchou bezeztrátovou kompresi založenou na algoritmu RLE. U PCX je jasně patrná nekoncepčnost návrhu a závislost prvních verzí tohoto formátu na použitých grafických kartách, resp. jejich videorežimech. [25]

Formát (hlavička souboru i uložení rastrových dat) kopíroval možnosti tehdejších grafických karet, tj. zejména značně omezený počet barev, organizaci dat do bitových rovin, malou barevnou paletu apod. S dalším rozvojem grafických schopností počítačů PC docházelo k nárůstu složitosti celého formátu PCX a různým přídatným řešením pro určité případy, např. pro 8 bitový obrázek to bylo umístění barevné palety nikoli přímo do hlavičky, ale na samotný konec souboru. Z těchto důvodů lze grafický formát PCX považovat spíše za odstrašující příklad (jak se nemá navrhovat souborový formát). [25]

Grafický formát PCX existuje v několika verzích. Tyto verze se chronologicky odvozují od verzí programu PC-PaintBrush a od jeho grafických schopností. V prvních verzích PC-PaintBrush podporoval pouze grafické karty CGA (černo-bílé a čtyřbarevné obrázky), posléze se přidávala podpora pro další grafické karty (EGA, VGA, SVGA) až do chvíle, kdy byl program portován na operační systém Microsoft Windows, který ovládání grafických karet kompletně převzal a aplikace na nich přestala být závislá. Jednotlivé dostupné verze formátu PCX jsou vidět v *Tab.9*. Většina grafických konvertorů vytváří obrázky s označením verze 5. Verze jsou do značné míry zpětně kompatibilní. [25]

Tab.9 Verze formátu PCX [25]

Verze PCX	Verze PaintBrushe	Možnosti
0	2.5	ukládané obrazy odpovídají možnostem CGA
2	2.8	šestnáctibarevné obrazy s barevnou paletou
3	2.8	šestnáctibarevné obrazy, ale bez barevné palety
4	pro Windows	truecolor grafika, podpora sekundární palety
5	3.0+	truecolor grafika, podpora sekundární palety

Grafický soubor typu PCX obsahuje hlavičku, která má vždy délku 128 Bytů. Využito je však pouze 70 Bytů, zbývajících 58 Bytů může využít aplikace, ovšem s tím, že se jejich obsah může kdykoliv, například po konverzi a/nebo editaci přepsat. Hlavička je tedy využita pouze minimálně. Provedení hlavičky formátu PCX je možné zhlédnout v *Tab.10*. [25]

Tab.10 Hlavička formátu PCX [25]

Offset	Velikost [Byte]	Význam
0	1	identifikace - vždy obsahuje číslo 0Ah
1	1	číslo verze (viz <i>Tab.9</i> )
2	1	komprese - 0-žádné ; 1-RLE
3	1	počet bitů na pixel v jedné obrazové rovině
4	8	souřadnice obrazu X1, Y1, X2, Y2 (little endian)
12	2	horizontální rozlišení
14	2	vertikální rozlišení
16	48	barevná paleta o rozsahu max. 16 barev
64	1	rezervovaná položka
65	1	počet obrazových rovin
66	2	počet Bytů na obrazový řádek
68	2	způsob interpretace palety: 0-barvy ; 1-odstíny šedi
70	58	nepoužito - možné využití aplikací
127	...	počátek obrazových dat

Celkový počet bitů na pixel se vypočte tak, že se vynásobí počet bitů na pixel v jedné obrazové rovině (offset 3) a počet obrazových rovin (offset 65). Možnosti kombinací hodnot těchto dvou položek znázorňuje *Tab.11*. [25]

Tab.11 Povolené kombinace hodnot počtu bitů na pixel a bitových rovin PCX [25]

Typ obrazu	Počet bitů na pixel	Počet obrazových rovin
CGA, černo-bílý (1 bit)	1	1
CGA, 4 barvy (2 bity)	2	1
EGA, 8 barev (3 bity)	1	3
EGA, VGA, 16 barev (4 bity)	1	4
VGA, 16 barev (4 bity)	1	4
VGA, 16 barev (4 bity)	4	1
VGA, 256 barev (8 bitů)	8	1
SVGA, truecolor (24 bitů)	8	3

Z Tab.11 lze vyčíst, že obrázky s maximálně šestnácti barvami mohou být uloženy dvěma odlišnými způsoby: [25]

- První způsob spočívá ve využití jedné obrazové roviny a čtyř bitů na pixel.
- Druhý způsob naopak používá čtveřici obrazových rovin a v každé obrazové rovině je pro jeden pixel rezervován pouze jeden bit - tento způsob více odpovídá způsobu práce s šestnáctibarevnou grafikou na PC, ale poněkud snižuje kompresní poměr.

Pokud jsou použity čtyři obrazové roviny, je způsob jejich ukládání následující: [25]

1.obrazová rovina 1.řádku

2.obrazová rovina 1.řádku

3.obrazová rovina 1.řádku

4.obrazová rovina 1.řádku

1.obrazová rovina 2.řádku

2.obrazová rovina 2.řádku

3.obrazová rovina 2.řádku

4.obrazová rovina 2.řádku

.....atd.....

U osmi bitových obrázků je problém s barevnou paletou. Zde se v plné míře ukazuje nedostatečně promyšlený návrh formátu PCX, protože v hlavičce souboru je rezervováno

místo pouze pro 16 barev palety. Musel se tedy najít jiný způsob uložení. Ten spočívá v tom, že je barevná paleta uložena na konci souboru. Test, zda je zde grafická paleta skutečně uložena, spočívá v načtení 769 Bytů od konce souboru. Pokud tento Byte obsahuje hodnotu C0h, je barevná paleta přítomna a zbylých 768 Bytů obsahuje 256 trojic hodnot RGB. [25]

Existují i obrázky, které mají barevnou paletu kratší, typicky 64 nebo 128 položek. Je tedy nutné testovat i Byte 193 ( $64 \times 3 + 1$ ) a 385 ( $128 \times 3 + 1$ ) od konce souboru. Co se stane v případě, že paleta není přítomna a náhodou je zrovna na pozicích 193, 385 či 769 od konce souboru uložena hodnota C0h, specifikace PCX neřeší, takže je ještě zapotřebí při načítání obrázku provádět testy, kolik rastrových dat se opravdu načetlo a zda po načtení celého rastru zbyvá v souboru 193, 385 nebo 769 Bytů. [25]

True Color (24 bitové) obrázky jsou ukládány do tří obrazových rovin. V každé obrazové rovině je rezervováno osm bitů na pixel. Obrazové roviny jsou ukládány prokládaně: [25]

1.obrazová rovina (R) 1.řádek

2.obrazová rovina (G), 1.řádek

3.obrazová rovina (B), 1.řádek

1.obrazová rovina (R), 2.řádek

2.obrazová rovina (G), 2.řádek

3.obrazová rovina (B), 2.řádek

1.obrazová rovina (R), 3.řádek

.....atd.....

Rastrová data mohou být v grafickém formátu PCX uložena buď v přímé (tj. nekomprimované) podobě, nebo v komprimovaném tvaru. PCX se bez komprese prakticky nepoužívají, většina obrázků uložených v tomto formátu používá jedinou podporovanou kompresní metodu - modifikovaný algoritmus RLE. [26]

RLE použitý u PCX pracuje s proudem Bytů, přičemž maximální délka tohoto proudu odpovídá délce obrazového řádku (tato hodnota je uložena v hlavičce). Obrazový řádek se postupně načítá a zjišťuje se, kolik Bytů (nikoli pixelů!) má stejnou hodnotu. Blok za sebou jdoucích Bytů se stejnou hodnotou se zapíše jako dvojice Bytů: první Byte udává

počet znaků v bloku, druhý Byte hodnotu těchto Bytů. Počítadlo Bytů je inicializováno na hodnotu C0h (nejvyšší dva bity jsou 1), to znamená, že pokud dekomprimační program narazí na Byte větší než C0h, tak ví, že se jedná o komprimovaný blok se dvěma Byty. [26]

Jednotlivé Byty, které nejsou součástí bloku a mají hodnotu menší než C0h, jsou do komprimovaného souboru zapsány ve své původní podobě. Horší je to s Byty, které mají hodnotu větší nebo rovno C0h. Aby nenastala kolize s počítadlem, musí se tyto Byty uložit jako dvojice Bytů C1h C0h (popř. jiná hodnota v intervalu <C0h ; FFh >), tj. jako blok o délce jednoho Bytu s barvou pixelu uloženou ve druhém Bytu. V tomto případě tedy nenastává komprese, ale naopak prodloužení (expanze) výstupního souboru. To vede k zajímavému paradoxu, který se u jiných kompresních metod neprojevuje: pouhou změnou indexu Bytů a úpravou barevné palety je možné měnit kompresní poměr u PCX souborů (ideální je, aby paleta byla seříděna tak, že nejčastěji používané barvy jsou uloženy na začátku, aby se omezil počet Bytů s hodnotou větší nebo rovno C0h). Kupodivu velmi málo aplikací tuto zajímavou optimalizaci provádí. [26]

#### 4.4 PNG

PNG (Portable Network Graphic Format) byl navržen s cílem vytvořit jednoduchý formát, který se dá snadno implementovat, který je plně přenositelný a který dosahuje nebo překračuje všechny schopnosti formátu GIF. Bylo také nutné, aby byl PNG volně dostupný a nezatížený licenčními poplatky a patentovými spory. [3]

PNG umožňuje ukládat 1 až 48 bitů na pixel. Konkrétně pak u odstínů šedi až 16 bitů a v systému pravých barev až 48 bitů na pixel a až 16 bitů dat alfa. Pro typ obrázku s odstíny šedi platí, že hodnota každého pixelu vyjadřuje procentuální světlost mezi 0 % (zcela černá) a 100 % (čistě bílá). V PNG je umožněno progresivní zobrazení a uchovávání hodnoty gama, údaje o průhlednosti a textové informace. [3]

Podle typu ukládaného obrázku je možné buď přímo každému pixelu umístěnému v rastrové mřížce nebo každé barvě uložené v barevné paletě přiřadit hodnotu průhlednosti. Podporován je jak plný osmibitový či dokonce šestnáctibitový alfa kanál (256 či 65536 stupňů průhlednosti, tj. hodnot alfa), tak i jednobitový. [27]

Maximální velikost předlohy formátu PNG může být až 2G×2G pixelů. Jako kompresní algoritmus je v tomto formátu využita komprese LZ77 a huffmanovo kódování. Tento

způsob komprese se také nazývá Deflate. [3] Posuvné okno u této komprese může mít nanejvýš 32768 Bytů [28].

Grafický formát PNG obsahuje jednu velmi prospěšnou funkci - na každý obrazový řádek je možné ještě před jeho kompresí aplikovat jednoduchý konvoluční filtr, jehož úkolem je připravit data tak, aby se komprimovala lépe. Většinou se počítá s tím, že se po aplikaci filtru (který většinou hledá podobnosti sousedních pixelů) na data objeví větší množství pixelů s nulovou hodnotou, nebo hodnotou blízkou nule (nebo naopak hodnotě 255 [29]), čímž se statisticky zvýší pravděpodobnost toho, že komprimační program v datech nalezne delší shodné posloupnosti a tím zmenší výslednou délku souboru. [27]

Filtrů, které je na jednotlivé obrazové řádky možné aplikovat, existuje v PNG celkem 5 [29]. Každý filtr musí provádět operaci, ke které lze nalézt operaci inverzní. Filtry jsou aplikovány na Byty, nikoli pixely. V *Tab.12* jsou vidět jednotlivé funkce filtrů. [28] Tyto Byty reprezentují buď odstín šedi či index do barevné palety nebo hodnotu uloženou v jednom barevném kanálu (obrázky typu RGB a RGBA). [29]

*Tab.12 Typy filtrů formátu PNG [28]*

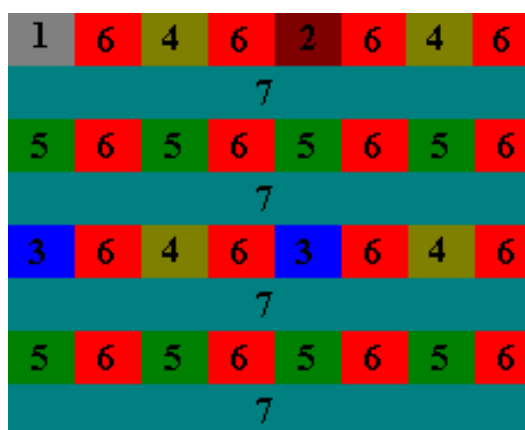
Číslo filtru	Jméno	Funkce filtru	Reverzní funkce
0	None	$Filt(x) = Orig(x)$	$Recon(x) = Filt(x)$
1	Sub	$Filt(x) = Orig(x) - Orig(a)$	$Recon(x) = Filt(x) + Recon(a)$
2	Up	$Filt(x) = Orig(x) - Orig(b)$	$Recon(x) = Filt(x) + Recon(b)$
3	Average	$Filt(x) = Orig(x) - \text{Floor}((Orig(a) + Orig(b)) / 2)$	$Recon(x) = Filt(x) + \text{floor}((Recon(a) + Recon(b)) / 2)$
4	Paeth	$Filt(x) = Orig(x) - \text{PaethPredictor}(Orig(a), Orig(b), Orig(c))$	$Recon(x) = Filt(x) + \text{PaethPredictor}(Recon(a), Recon(b), Recon(c))$

Nultý filtr žádnou filtraci neprovádí - pouze provádí kopii vstupních pixelů do pixelů výstupních bez jakéhokoli zpracování. Filtr číslo 1 má název Sub. Jedná se o velmi jednoduchý filtr, který na svůj výstup posílá rozdíl mezi dvěma Byty. Filtr číslo 2 má název Up. Jedná se o obdobu filtru Sub, pouze s tím rozdílem, že se porovnávají barevné hodnoty pixelů umístěné nad sebou a nikoli vedle sebe. [29]

Filtr číslo 3 s názvem Average vznikl kombinací předchozích dvou typů filtrů. Už se nebere v úvahu barva pouze jednoho sousedního pixelu, ale hned pixelů dvou - horního a levého souseda. Tento filtr nejhůře reaguje na hrany o úhlu 45°. [29]

Posledním a nejsložitějším typem filtru je filtr číslo 4 - Paethův filtr. Nejedná se o klasický konvoluční filtr, protože obsahuje prediktor, což je funkce, která na základě hodnot tří sousedních pixelů (vlevo, nahoře a vlevo nahoře) vybere ten, jehož barva je nejbližší pixelu zpracovávanému. [29]

Zatímco u formátu GIF je prokládání nesymetrické, protože je prováděno pouze ve vertikálním směru, u PNG je použito zcela symetrické prokládání, tj. pixely jsou ukládány „napřeskáčku“ jak ve směru vertikálním, tak i ve směru horizontálním. Symetrický způsob přináší tu výhodu, že prvotní náhled na obrázek je možné provést již po přenesení pouhé 1/64 všech pixelů (u formátu GIF 1/8). Způsob zobrazení prokládaného obrázku s rozlišením 8×8 pixelů je vidět na *Obr.30* - čísla značí pořadí průchodu. Je nutné poznamenat, že změna posloupnosti ukládaných pixelů má negativní vliv na kompresní poměr, protože se statisticky zmenšuje korelace mezi sousedními hodnotami. [27]



*Obr.30 Prokládaný režim formátu PNG*

V grafickém formátu PNG je možné uchovat gamma faktor zařízení, které obrázek vytvořilo (skener, fotoaparát, počítačový program). Při zobrazování se na základě tohoto gamma faktoru mohou intenzity pixelů přepočítat a dosáhne se linearity změny světlosti zobrazovaných pixelů na různých zobrazovacích zařízeních. Důležité je, že gamma korekci je možné provádět beze změny hodnot pixelů původního obrázku, takže při úpravách a přenosu na další platformu nedochází ke ztrátě dynamiky barev. [27]

Pro profesionální barevný tisk však samotná informace o gamma korekci nemusí být dostačující, protože každý barvový kanál (RGB) může vykazovat jinou nelinearitu, která

může vést buď ke ztrátě dynamiky barev nebo dokonce k celkovému rozpadu spektra. Zde přichází ke slovu změna barevnosti prováděná v barevném prostoru x-y definovaném konsorciem CIE a kompletní barevné profily ICC. Oboje je v grafickém formátu PNG podporováno. Plný barevný profil ICC je spíše výjimkou a mnohdy i zbytečným luxusem, který zvětšuje celkovou velikost souboru i o stovky Bytů. [27]

Hlavička PNG má délku pouhých osmi Bytů, které mají vždy konstantní hodnoty, a to: [30]  
89h 50h 4Eh 47h 0Dh 0Ah 1Ah 0Ah.

Význam jednotlivých Bytů v hlavičce je možné vidět v *Tab.13*. [30]

*Tab.13 Hlavička souboru formátu PNG [30]*

Byte	Význam
89h	jedná se o Byte s nastaveným nejvyšším bitem (detekce sedmibitového přenosu)
50h 4Eh 47h	řetězec ‚PNG‘, spolu s prvním Bytem jednoznačně detekuje typ souboru
0Dh 0Ah	znaky CR a LF, detekce náhrady za jinou sekvenci: CR, LF či LF a CR
1Ah	znak Ctrl+Z, pro příkaz typu MS-DOSu
0Ah	znak LF, detekce náhrady za CR+LF či LF+CR

V osmi Bytech se tvůrcům PNG podařilo vytvořit sekvenci Bytů, na kterých je možné otestovat, zda přenos proběhl v pořádku. Hned první Byte byl zvolen tak, aby byl větší než 7Fh, tj. aby měl nastavený nejvyšší bit na jedničku. To má dva významy: detekuje se defektní sedmibitový přenos a některé textové editory soubor správně detekují jako binární a ne ASCII. [30]

Dále následují tři znaky tvořící řetězec ‚PNG‘, což způsobuje, že hlavička souboru je částečně čitelná při zběžném pohledu i pro člověka. Následuje dvojice znaků CR a LF, na kterých je možné detekovat problémy při přenosu souboru mezi různými platformami. [30]

Dále je znak Ctrl+Z, který při zadání příkazu: „type obrazek.png“ v prostředí MS-DOS/MS-Windows způsobí výpis textu: „ëPNG“. tj. zobrazí se pouze čtyři znaky, za kterými je řádek i výpis ukončen. Nemůže tedy nastat situace, kdy se po zadání příkazu PNG obrazovka zaplní různými „paznaký“. Jediným vypsaným paznakem je ono ë (či jiný

znak v závislosti na lokalizaci), které by mělo uživatele varovat, že se pokouší vypsát obsah binárního souboru. [30]

Poslední znak v hlavičce je LF, který opět slouží pro detekci, zda nedošlo k jeho náhradě jiným znakem či jinou sekvencí znaků, typicky CR, CR+LF či LF+CR. Ihned za hlavičkou totiž následuje sekvence tří nulových Bytů, které by byly náhradou znaku CR za dva znaky posunuty, což opět způsobí detekovatelnou chybu při čtení. [30]

Veškeré informace i metainformace o zpracovávaném obrázku jsou uloženy v blocích dat nazvaných chunky (přibližný překlad je blok). Každý chunk se skládá ze čtyř částí: [30]

- První část má konstantní velikost čtyři Byty a obsahuje celkovou délku datové části chunku. Teoreticky je tedy maximální délka datové části rovna  $2^{32}-1$  Bytům; avšak pro snazší implementaci, například v těch programovacích jazycích, které nemají implementovaný beznaménkový datový typ (Java), je maximální hodnota délky rovna  $2^{31}-1$  Bytů. V praxi jsou však délky chunků o několik řádů menší.
- Druhá část chunku má opět velikost čtyři Byty. Obsahuje jméno (typ) chunku ve formátu čtyř ASCII znaků malé i velké anglické abecedy. Jedná se o ASCII kódy v rozsahu 65–90 a 97–122 decimálně. Jméno chunku může být načteno a následně rozpoznáno pouhou jednou operací porovnání (32 bitových integerů) bez nutnosti implementace řetězcového porovnání a současně je typ chunku velmi dobře čitelný i pro člověka. Velikost znaků ve jménu (minusky/verzálky) dále určuje, zda je daný chunk pro zpracování obrázku volitelný či povinný.
- Třetí část chunku je tvořena vlastními daty. Tato část má v některých případech, například u chunku nazvaného IDAT, nulovou délku.
- Poslední čtvrtá část chunku má délku čtyři Byty a obsahuje kontrolní součet (CRC) druhé a třetí části, tj. jména (typu) chunku a uložených dat. Vzhledem k tomu, že délka chunku není do kontrolního součtu zahrnuta, je možné CRC generovat přímo při zápisu dat bez nutnosti druhého průchodu v případě, že délka chunku není dopředu známá (například při kompresi). Použitý polynom má tvar:  
$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

Všechny vícebytové položky v chunkcích (včetně jejich délky) jsou uloženy v pořadí big endian, tj. Byte s nejvyšší vahou je v souboru na prvním místě. [30]

Jméno (resp. typ) chunku je uloženo ve čtyřech Bytech. Vždy se jedná o posloupnost čtyř ASCII znaků malé či velké abecedy, tzn. z celkového množství  $2^{32}$  jmen je jich možné použít  $(2 \times 26)^5 = 380\,204\,032$ . [30]

Vzhledem k tomu, že je jméno chunku tvořeno čtyřmi znaky, máme k dispozici celkem čtyři bity, kterými můžeme rozlišit různou důležitost chunků. Význam pátých bitů každého ze znaků názvu je následující: [30]

- Pátý bit prvního znaku názvu - tento bit určuje, zda je daný chunk pro zpracování obrázku nezbytný. Příkladem chunku, který je nutné vždy zpracovat, je IHDR, mezi nepovinný chunk patří tEXt.
- Pátý bit druhého znaku názvu: tento bit určuje, zda je typ chunku veřejný (určený specifikací a známý více aplikacím) nebo privátní, tj. dávající smysl pouze jedné či několika málo aplikacím. Díky tomuto bitu nikdy nenastanou kolize mezi oficiálními a aplikačními chunky.
- Pátý bit třetího znaku názvu: je ve verzi PNG 1.0, PNG 1.1 i PNG 1.2 vždy vynulován, tj. třetí znak názvu chunku je vždy zapsán velkým písmenem.
- Pátý bit čtvrtého znaku názvu: tento bit je určen zejména pro grafické editory a konvertory obrázků. Jeho hodnota určuje, zda se při změně obrázku (editaci) má daný chunk uložit zpět do souboru, či zda se se změnou obrázku chunk stane nepoužitelným. Příkladem může být chunk obsahující histogram - při změně obrázku dojde i ke změně histogramu, a proto není korektní provést kopii chunku ze zdrojového obrázku do obrázku cílového.

Každý obrázek typu PNG obsahuje tři povinné chunky v následujícím pořadí: [30]

- IHDR - hlavička obrázku
- IDAT - datová část (pixmap)
- IEND - ukončující značka

U obrázků obsahujících barevnou paletu přibývá ještě jeden typ chunku s názvem PLTE a je umístěn mezi chunk IHDR a IDAT. [30] Barevná paleta se nachází ještě před datovým chunkem IDAT z toho důvodu, že některé prohlížeče obrázek zobrazují již při jeho přenosu

po mnohdy pomalých datových linkách a bez informace o barvové paletě není možné obrázek průběžně korektně zobrazit. [31]

V chunku typu IHDR je uložena hlavička obrázku. Ta je odlišná od hlavičky celého souboru uvedené v předchozím textu. Hlavička obrázku obsahuje základní metainformace o uloženém obrázku, zejména jeho rozlišení, bitovou hloubku, typ kódování barev a použitou filtraci. Vzhledem k tomu, že datová část hlavičky obrázku má vždy délku 13 Bytů, začíná chunk posloupností 00h 00h 00h 0Dh. Hlavička obrázku musí být uvedena ihned za hlavičkou PNG, při čtení je tedy nutné otestovat, zda se ihned po přečtení hlavičky PNG objeví tato posloupnost. Pokud tomu tak není, mohlo dojít k poškození souboru. [30]

Datová část chunku IHDR obsahuje položky uvedené v *Tab.14*. [30]

*Tab.14 Datová část chunku IHDR formátu PNG [30]*

Offset	Velikost [Byte]	Význam
0	4	šířka obrázku uvedená v pixelech
4	4	výška obrázku uvedená v pixelech
8	1	bitová hloubka pixelů v barvovém kanálu (povolené hodnoty jsou 1, 2, 4, 8 a 16)
9	1	typ kódování barev
10	1	použitá metoda komprese (vždy 0 - Deflate)
11	1	použitá metoda filtrace (vždy 0 - adaptivní filtrace)
12	1	prokládání obrázku (0-bez prokládání, 1-prokládání)

Položka udávající typ kódování barev může nabývat hodnot uvedených v *Tab.15*. [30]

*Tab.15 Typ kódování barev formátu PNG [30]*

hodnota	bitová hloubka	Význam
0	1, 2, 4, 8, 16	Obrázek v odstínech šedi
2	8, 16	Plnobarevný (true color) obrázek typu RGB
3	1, 2, 4, 8	Obrázek s barevnou paletou (musí existovat chunk PLTE)
4	8, 16	Obrázek v odstínech šedi a s průhledností (alfa kanálem)
6	8, 16	Obrázek, kde každý pixel obsahuje hodnoty RGBA

Chunk IDAT obsahuje 4 Byty s délkou chunku a dále následují hodnoty 49h 44h 41h 54h, které dávají řetězec „IDAT“. Poté následují zkomprimovaná data. Jeden komprimovaný datastream je u PNG uložen v „zlib“ formátu, jehož struktura je vidět v tab *Tab.16*. [28]

*Tab.16 Struktura formátu ZLIB [28]*

Velikost [Byte]	Význam
1	zlib kompresní metoda / kódy příznaků
1	doplňující příznaky / kontrolní bity
-	zkomprimované bloky dat
4	kontrolní hodnota

Zkomprimované data uvnitř datastreamu jsou uložena jako série bloků, z nich každý může představovat nekomprimovaná data, data komprimovaná LZ77 s fixním Huffmanovým kódováním, nebo data komprimovaná LZ77 s Huffmanovým kódováním přizpůsobeným datům. Kompletní filtrovaný obrázek formátu PNG je reprezentován jedním zlib datastreamem, který je uložen v několika IDAT chunkích. [28]

Kontrolní hodnota je vypočtena z původních dat bez komprese. Tuto hodnotu není možné zaměňovat s CRC kódem - výpočet se od výpočtu CRC kódu liší. [28]

Chunk IEND by se měl nacházet na konci každého PNG souboru, aby byl program informován, že může ukončit načítání a že se soubor byl přenesen celý. Tento chunk neobsahuje žádnou datovou část, proto je jeho Bytová sekvence o délce dvanácti Bytů vždy stejná. Konkrétní hodnoty jsou uvedeny v *Tab.17*. [30]

*Tab.17 Chunk IEND formátu PNG [30]*

Sekvence Bytů	Význam
00h 00h 00h 00h	Délka datové části chunku: 0
49h 45h 4Eh 44h	Sekvence ASCII znaků „IEND“
AEh 42h 60h 82h	CRC tohoto chunku

Soupis některých významných nepovinných chunků, které se mohou v grafických souborech PNG vyskytnout je možné vidět v *Tab.18*.

Tab.18 Významné nepovinné chunky formátu PNG [30]

Typ chunku	Název chunku
gAMA	Gamma Value
bKGD	Background Color
iTXt	International Text Annotations
hIST	Histogram
sPLT	Suggested Palette
sBIT	Significant Bits
pHYs	Physical Pixel Dimensions
sCAL	Physical Scale
oFFs	Image Offset
pCAL	Pixel Calibration
fRAc	Fractal Parameters

## 4.5 TGA

Grafický formát TGA (Targa) byl navržen firmou Truevision. Tento Formát je mezi programátory oblíben zejména díky velmi jednoduché a snadno dekódovatelné struktuře. Historicky se také jednalo o jeden z prvních grafických formátů, který i na obyčejná PC přinesl true color grafiku. [32]

Formát TGA je možné použít jak pro rastrové obrázky s barevnou paletou, tak i pro obrázky uložené ve stupních (resp. odstínech) šedi či obrázky typu true color ( $2^{24}$  barev). Podporován je i až osmibitový alfa kanál. [32]

V grafickém formátu typu TGA je možné ukládat bitmapy různých typů. Pravděpodobně nejpoužívanější je nekomprimovaná bitmapa uložená v pravých barvách (true color), je však možné uložit i bitmapu ve stupních šedi či bitmapu obsahující místo přímých barev indexy do barvové palety. [32]

TGA je možné komprimovat metodou RLE [3]. Podle [32] může být kódování RLE kombinované s Huffmanovým kódováním. Specifikace formátu TGA [33] ovšem uvádí pouze kompresi pomocí RLE, proto také není divu, že [32] dále dodává: „Mnoho převodních či prohlížečích programů podporují pouze variantu RLE bez Huffmanova kódování“.

Specialitou formátu TGA je schopnost ukládat obrázky v osmibitové barevné hloubce, ovšem bez barevné palety. Toho se využívá při práci s obrázky uloženými ve stupních šedi. Převod na snímky s paletou je pro zpracování obrazu v tomto případě zcela nevhodný.

V počítačové grafice se jedná například o naskenované obrázky nebo textury s luminancí bez přidané barevné informace. Příkladem použití jsou statické světelné mapy. [32]

Všechny informace jsou v souborech typu TGA rozděleny do čtyř sekcí, přičemž pouze první sekce je povinná. Sekce nemusí obsahovat identifikační hlavičku - pozice sekcí v souboru je možné zjistit již po načtení informační hlavičky souboru. [32]

V první sekci umístěné na začátku souboru je uložena informační hlavička, jejíž velikost je vždy rovna 18 Bytům. V této hlavičce jsou specifikovány všechny důležité informace o rastrovém obraze a způsobu jeho uložení v souboru. Struktura hlavičky formátu TGA lze vidět v *Tab.19*. [32]

*Tab.19 Hlavička formátu TGA [32]*

Offset	Velikost [Byte]	Význam
0	1	velikost identifikačního pole
1	1	typ barevné palety - 0-neobsažena ; 1-obsažena
2	1	typ obrázku
3	2	počátek barevné palety
5	2	délka barevné palety
7	1	bitová hloubka položek barevné palety
8	2	X-ová souřadnice počátku obrázku (levý spodní roh)
10	2	Y-ová souřadnice počátku obrázku (levý spodní roh)
12	2	šířka obrázku uvedená v pixelech
14	2	výška obrázku uvedená v pixelech
16	1	počet bitů na jeden pixel (bitová hloubka)
17	1	popisovač obrázku

Hodnota udávající typ obrázku obsahuje informace o formátu uložení a kódování rastrových dat. Hodnoty, které zde mohou být uloženy a jejich význam je vidět v *Tab.20*. [32] K této tabulce je potřeba dodat, že ve specifikaci formátu TGA [33] se hodnoty 32 a 33 nevyskytují.

Tab.20 Povolené hodnoty v položce „typ obrázku“ formátu TGA [32]

Hodnota	Význam
0	žádná rastrová data nejsou uložena
1	nekomprimovaná data s barevnou paletou
2	nekomprimovaná data ve formátu RGB
3	nekomprimovaná data v odstínech šedi
9	data kódovaná RLE s barevnou paletou
10	data kódovaná RLE ve formátu RGB
11	data kódovaná RLE v odstínech šedi
32	data kódovaná Huffmanovým kódem a RLE s barevnou paletou
33	data kódovaná Huffmanovým kódem a RLE s barevnou paletou (uložení v quadtree)

Podporovaný počet bitů na pixel tohoto formátu je 8, 16, 24, 32 [3]. Podle [32] je podporována také 1 bitová hloubka. Ovšem podle specifikace formátu TGA [33] toto není pravda. Na webové stránce [34] je zmíněno, že se jedná o ideové rozšíření původních typů obrázků TGA, které ovšem nebylo touto firmou oficiálně schváleno, a proto je vhodnější pro uložení obrázku s touto barevnou hloubkou použít jiný formát.

V položce popisovač obrázku jsou uloženy příznaky (tzv. flags) specifikující informace o posloupnosti uložených pixelů. Jedná se o počet atributových (alfa) bitů na pixel a o údaj s informací, ke kterému rohu obrazovky se váže počátek obrazových dat. [32] Zdroj [32] dále uvádí, že 6-7 bit značí způsob prokládání řádků (00-žádné, 01-liché/sudé, 10-čtyři řádky, 11-rezervováno). Specifikace formátu TGA [33] však tyto bity označuje jako rezervované a vždy s hodnotou 0. Jedná se tedy o nestandardní využívání těchto bitů.

Za informační hlavičkou může následovat identifikační pole obrázku, což je textový řetězec o maximální délce 255 znaků. Do tohoto pole lze ukládat libovolné údaje. Při pohledu na způsob uložení identifikačního pole v souborech typu TGA lze vidět, že není vyřešeno k plné spokojenosti, protože se toto pole může kdykoli přepsat a aplikace se tedy nemohou spolehnout na jeho obsah. Tato sekce je nepovinná a v obrazových souborech se moc často nevyskytuje. [32]

Ve třetí sekci může být uložena barevná paleta. Tato sekce je nepovinná a používá se pouze u některých obrázků s formátem 8 bitů na pixel. Paleta obsahuje hodnoty barevných složek ve formátu RGB pro každou položku uloženou v paletě. Celkový počet položek barevné palety, její počátek a počet bitů rezervovaných na jednu položku lze zjistit

v hlavičce souboru TGA (viz *Tab.19*). Podle počtu bitů rezervovaných na jednu položku jsou povoleny tři formáty položek v barvové paletě: [32]

- 32 bitů: barevné složky RGB jsou spolu se složkou alfa (průhledností) uloženy za sebou v pořadí modrá, zelená, červená a alfa (RGBA). Každá složka má velikost jeden Byte. Opačné pořadí barevných složek (tj. BGRA) většinou není problémem, neboť odpovídá pořadí Bytů při čtení na procesorech Intel (formát TGA vznikl pro potřeby aplikací pracujících na těchto procesorech), nicméně aplikace, která aspiruje na přenositelnost, by měla barvové složky načítat po Bytech nebo používat makra, která automaticky provedou „přeskládání“ Bytů v rámci načteného 32bitového slova, podobně jako u grafického formátu BMP.
- 24 bitů: barevné složky RGB jsou za sebou uloženy v pořadí modrá, zelená a červená, tj. ve skutečnosti jde o formát BGR. Každá složka má velikost opět jeden Byte.
- 16 bitů: barevné složky RGB jsou spolu s příznakem průhlednosti (pouze 1 bit, tzn. 0 až 100 % průhlednosti) uloženy v bitové struktuře:  $AR_4R_3R_2R_1R_0G_4G_3G_2G_1G_0B_4B_3B_2B_1B_0$ .

V sekci čtvrté jsou uložena vlastní rastrová data, tj. barvy jednotlivých pixelů. Posloupnost rastrových dat (zejména orientaci vertikální osy) lze ve formátu TGA specifikovat přímo v hlavičce, je například možné obrázky ukládat od prvního řádku do řádku posledního či naopak. Jak již bylo zmíněno, rastrová data mohou být komprimována jednoduchým RLE algoritmem. [32]

Formát uložení rastrových dat je vždy typu little endian, tzn. v konvencích používaných u procesorů Intel. Pro nekomprimované obrázky uložené ve 8 bitové barevné hloubce je každý pixel v rastru reprezentován jedním Bytem. Pro nekomprimované obrázky uložené ve 16 bitové barevné hloubce (vždy bez barevné palety) je každý pixel uložen ve dvou Bytech, které mají bitovou strukturu  $AR_4R_3R_2R_1R_0G_4G_3G_2G_1G_0B_4B_3B_2B_1B_0$ . Nekomprimované obrázky v barevné hloubce 24 nebo 32 bitů jsou uloženy dle konvence procesorů Intel, tj. ve formátu BGR a BGRA. [32]

Při použití RLE kódování jsou posloupnosti po sobě následujících pixelů s barvami se stejnou hodnotou zakódovány dvojicí hodnot: počtem opakování a hodnotou opakování. V režimu kódování RLE je nejvyšší bit (tj. hodnota 80h) každého Bytu na začátku

datového paketu rezervován jako logický příznak, zda se bude jednat o opakování (RLE paket) či nikoliv. Pokud je tento příznak nastaven, tak udávají hodnoty nižších sedmi bitů počet opakování barvy, která je uložena v následujících Bytech - 1 Byte pro osmibitové obrázky, 2 Byte pro šestnáctibitové obrázky atd. [32]

Počet opakování se vypočte jako: [32]

$$\text{Opakování} = \text{hodnota} \& 0x7F + 1$$

V případě, že je příznak pro opakování 0 (tj. v prvním Byte posloupnosti je uložena hodnota menší než 80h), udávají hodnoty nižších sedmi bitů nekomprimovaná data [32].

RLE paket by nikdy neměl kódovat pixely z více než jednoho řádku - přestože tedy bude např. poslední pixel 4. řádku stejný jako první pixel 5. řádku, musí být tyto pixely zakódovány odděleně. Tímto se zajistí možnost přístupu na jakýkoliv řádek bez nutnosti dekódování veškerých obrazových dat. [33]

Na různých webových stránkách, např. [34] nebo [35] lze nalézt informaci, že je možné kódovat pixely různých řádků dohromady. Podle [36] obsahuje příloha C technické příručky Truevision se starou specifikací formátu TGA větu: „For the run length packet, the header is followed by a single color value, which is assumed to be repeated the number of times specified in the header. The packet may cross scan lines (begin on one line and end on the next)“. Z tohoto by skutečně vyplývalo, že některé obrázky formátu TGA mohou být takto kódovány. Novější specifikace [33] str.27 však jasně říká: „Run-length Packets should never encode pixels from more than one scan line“.

Rozšíření zavedená do nového TGA formátu (verze 2.0) se nazývají Developer Area (oblast určená vývojářům), Extension Area (rozšířená oblast) a TGA File Footer (pata souboru). Tato pole byla přidána do původního TGA formátu bez toho, aby byly provedeny jakékoli změny v hlavičce. Aplikace, které čtou pouze původní TGA soubory, by neměly mít žádné problémy se čtením nových TGA souborů. Pokud však vývojářská a rozšířená oblast obsahují některé informace, které mohou ovlivnit správné čtení nebo zobrazování předlohy, mohou určité problémy nastat. [3]

Pata má velikost 26 Bytů a je to vždy poslední blok informací v souboru TGA verze 2.0. Obsahuje celkem tři pole, a to offset rozšiřující (Extension) oblasti (4 Byty), offset vývojářské (Developer) oblasti (4 Byty) a signaturu TGA (18 Bytů). [3]

Signatura obsahuje identifikační signaturní řetězec. TGA formát neobsahuje v hlavičce pole, které by vyjadřovalo verzi daného formátu. Namísto toho verze 2.0 obsahuje patu, v níž je 16 znakový řetězec, jenž danou verzi souboru vyjadřuje. K určení verze TGA souboru je nutné přečíst bity 8-23 v patě. Pokud se zde nachází řetězce znaků TRUEVISION-XFILE, jde o verzi TGA souboru 2.0. Za polem se Signaturou následuje ASCII hodnota 2Eh (perioda) a ASCII hodnota 00h (NULL). Obě tato pole musí obsahovat správné informace pro daný soubor tak, aby byl rozeznán jako soubor verze 2.0. [3]

Prvním vstupem ve vývojářské oblasti je počet adresářových vstupů nebo tagů. Toto pole má velikost 2 Byty. Za počtem adresářových vstupů je série 10 Bytových tagů, pro každý vstup jeden. [3]

Ve vývojářské oblasti se nachází identifikační číslo tagu (2 Byty), offset umístění dat tagu (4 Byty) a velikost dat tagu v Bytech (4 Byty). Tagy jsou ve vývojářském adresáři vždy umístěny v souvislém bloku a nemusí být seřazeny podle svých čísel. [3]

Hodnoty identifikačního tagu 0-32767 jsou rezervovány pro potřeby vývojářů. Hodnoty 32767-65535 jsou rezervovány pro potřeby Truevision. [3]

Rozšiřující oblast může být brána jako druhá hlavička obsahující informace, které se nenacházejí v původní TGA hlavičce [3]. Její velikost ve verzi 2.0 je 495 Bytů. Mimo jiné zde může být uloženo jméno autora, komentář, údaje o času, údaje o softwaru, klíčová barva (tzn. průhledná barva resp. barva pozadí), poměr stran, gamma hodnota, atributy ovlivňující alfa kanál atd. [33] Nepoužité položky jsou nastaveny na nulu [3]. Podrobnější detaily o struktuře rozšiřující oblasti lze nalézt ve specifikaci [33].

Nový TGA formát souboru může dále obsahoval tři nové datové struktury. Jsou to: tabulka vzorkovacích řádků, předloha poštovní známky a tabulka barevných korekcí. V TGA souboru může být přítomna jen jedna z nich a počátek těchto struktur je zapsán v rozšiřující oblasti. [3]

Tabulka vzorkovacích řádků je jednou z metod přístupu ke vzorkovacím řádkům ať už jsou na jakékoli pozici v hrubé nebo komprimované podobě. Tabulku si můžeme představit jako pole 4 Bytových hodnot. Každá hodnota představuje offset od začátku souboru k začátku daného řádku v datech předlohy. Každému řádku předlohy přísluší jeden vstup do tabulky. Vstupy jsou do tabulky zapsány v tom pořadí v jakém se řádky objevují v předloze. [3]

Předloha poštovní známky je menší zobrazení základní předlohy uložené v TGA souboru. První Byte dat známky udává šířku známky, druhý výšku známky v pixelech. Znamka by neměla být větší než 64×64 pixelů. Většinou je uložena ve stejném formátu jako základní předloha, ovšem není nikdy komprimována. [3]

Tabulka barevných korekcí je podle [33] blok dlouhý  $256 \times 4$  (hodnoty A, R, G, B)  $\times 2$  (velikost jedné hodnoty v Bytech) - dohromady tedy 2048 Bytů a ne 1000, jak uvádí [3]. Toto umožní uživateli uložit tabulku barevných korekcí pro barevné přemapování [33].

## 4.6 TIFF

Formát TIFF (Tag Image File Format) je asi nejvšestrannější a nejrozmanitější existující bitmapový formát. Jeho schopnost rozšíření a podpora mnoha datových kompresních schémat dovoluje vývojářům přizpůsobit si formát TIFF svým potřebám. [3]

Hlavním zdrojem problémů souborů TIFF spočívá v neschopnosti programů číst data nezávisle na jejich Bytovém uspořádání - tzn. čtení dat je provedeno jen v uspořádání malý endian nebo naopak jen v uspořádání velký endian. [3]

Jeden z dalších častých problémů je spojen s kompresí obrazových dat. Některé programy nemají implementovány všechny kompresní algoritmy a v dalších mohou být vytvořeny nestandardní kompresní algoritmy, které se původně ve specifikaci nevyskytují. [3] [37]

Soubory TIFF jsou organizovány do tří sekcí: [3]

- Hlavička souboru předlohy (Image File Header - IFH)
- Adresář souboru předlohy (Image File Directory - IFD)
- Bitmapová data.

V souboru musí být alespoň IFD a IFH. Je tedy možné narazit na soubor TIFF, který neobsahuje žádná data předlohy. Soubor TIFF, který obsahuje několik předloh má jeden IFD a jednu bitmapu na každou předlohu. [3]

Hlavička souboru předlohy formátu TIFF se skládá ze tří informačních polí a celkem má velikost 8 Bytů. Položky v hlavičce souboru a jejich význam jsou vidět v *Tab.21*. [3]

Tab.21 Hlavička formátu TIFF [3]

Offset	Velikost [Byte]	Význam
0	2	identifikátor řazení Bytů
2	2	číslo verze - resp. identifikátor TIFF (vždy 2Ah)
4	4	offset prvního obrazového adresáře (IFD)

Identifikátor obsahuje buď hodnotu 4949h („II“) nebo 4D4Dh („MM“). Tyto hodnoty vyjadřují, zda jsou data v souboru TIFF zapsána v uspořádání malý endian (4949h [38]) nebo v uspořádání velký endian (4D4Dh [38]). Všechna data za těmito dvěma Byty jsou v určeném Bytovém uspořádání. Tyto dvě hodnoty nebyly vybrány náhodně - mají stejný tvar nezávisle na celkovém Bytovém uspořádání souboru. [3]

Jestli je daný soubor skutečně ve formátu TIFF, lze zjistit podle prvních čtyř Bytů. Pokud to jsou 49h 49h 2Ah 00h nebo 4Dh 4Dh 00h 2Ah jde o soubor TIFF. [3]

IFD je blok informací podobný hlavičce u ostatních formátů. Jeho úkolem je popisovat data, se kterými souvisí. Obsahuje informace o výšce, šířce a hloubce předlohy, počet barevných ploch a typ datové komprese použité v bitmapových datech. Na rozdíl od většiny pevně daných hlaviček je IFD dynamická struktura, kde se nemusí měnit pouze její velikost, ale i její pozice v souboru. [3]

IFD může mít různou velikost, protože může obsahovat různý počet datových záznamů - tagů. Každý tag obsahuje jedinečný blok informací. Může nastat situace, kdy dvě IFD obsahují stejné tagy, ale tagy nejsou ve stejném pořadí. Struktura IFD je vidět v Tab.22. [3]

Tab.22 IFD formátu TIFF [3]

Offset	Velikost [Byte]	Význam
0	2	počet tagů v IFD
2	12 * počet_tagů	pole tagů
12 * počet_tagů + 2	4	offset k dalšímu IFD nebo hodnota 00h

Tag je 12 Bytový záznam, který obsahuje specifické informace o bitmapových datech. Tag může obsahovat libovolný typ dat a specifikace TIFF definuje přes 70 tagů. Tagy, které jsou definovány specifikací TIFF, se nazývají veřejné tagy a nesmějí být modifikovány jinak, než jak je povoleno parametry v poslední specifikaci. Tag může obsahovat nebo

ukazovat na data o libovolné velikosti a může být také umístěn kdekoli v IFD. Negativem této univerzálnosti je velikost tagu. Struktura tagu je vidět v *Tab.23*. Poslední čtyři Byty tagu mohou obsahovat kromě údaje o offsetu i vlastní data. [3]

*Tab.23 Tag formátu TIFF [3]*

Offset	Velikost [Byte]	Význam
0	2	identifikátor tagu
2	2	skalární typ datových položek
4	4	počet položek v datech tagu
8	4	Bytový offset k datovým položkám

Aby vývojáři nemuseli přemýšlet, který tag by měl být zapsán do souboru TIFF a které tagy je důležité číst, definuje specifikace koncept základních TIFF předloh. Tyto základy jsou definovány typem uložených dat: dvouúrovňové, odstíny šedi, paletově barevné a plně barevné. Každý základ má minimální sadu tagů, které se musí objevit v každém typu formátu TIFF. Typické tagy, které se ve formátu TIFF vždy objevují jsou ImageWidth, ImageLength, BitsPerSample, Compression, SamplesPerPixel atd. [3]

Hodnoty, které mohou být uloženy v tagu Compression (komprese) jsou vidět v *Tab.24* [38]. Další hodnoty tagu, které se používají v knihovně libtiff znázorňuje *Tab.25*. K této tabulce je nutné dodat, že Deflate komprese je považována za experimentální a není podporována všemi aplikacemi, proto je pro přenos obrázků vhodné použít jinou kompresi (např. LZW). [37]

*Tab.24 Hodnoty tagu Compression dle specifikace [38]*

Hodnota	Typ komprese
1	bez komprese
2	CCITT G31D bez instrukcí EOL a RTC
3	CCITT Group 3 (v tagu T4Options se dá nastavit typ 1D nebo 2D)
4	CCITT Group 4
5	LZW
6	TIFF JPEG 6.0 (zastaralá verze [37])
32773	PackBits

Tab.25 Doplnující hodnoty tagu Compression dle dokumentace knihovny libtiff [37]

Hodnota	Typ komprese
7	JPEG (nová verze)
32766	2-bitové kódovací schéma používané NeXTem
32771	CCITT
32809	4-bitové RLE schéma z ThunderScanu
32909	kompresní schéma Pixaru pro barevné obrázky s vysokým rozlišením
34676 a 34677	kompresní schéma SGI pro barevné obrázky s vysokým rozlišením
32946	Deflate (ZIP)

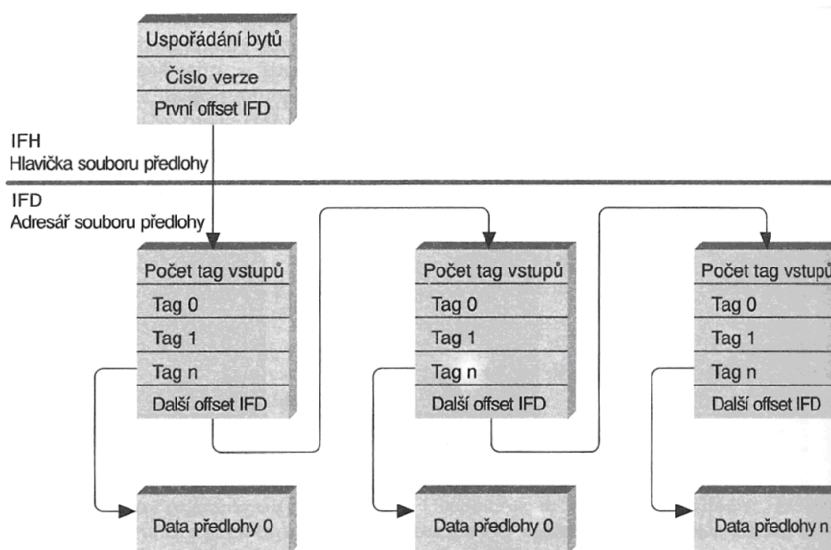
TIFF je považován za komplikovaný formát mimo jiné také z toho důvodu, že umístění každého IFD a dat, na která IFD ukazuje (včetně IFD dat), se může měnit. Jediná část souboru TIFF, která má pevnou pozici je IFH, kterému patří prvních osm Bytů každého souboru. Všechna ostatní data v souboru lze nalézt pomocí informací v IFD. Každý IFD a příslušná bitmapa jsou známy jako podsoubory TIFF (jejich počet není omezen). [3]

Obr.31 ukazuje tři možná uspořádání vnitřní datové struktury souboru TIFF obsahujícího tři předlohy. Ve všech případech se v souboru objevuje jako první IFH. V případě prvním byl každý z IFD zapsán do souboru před bitmapovými daty. Toto uspořádání je nejučinnější pro rychlé čtení IFD dat. V druhém případě se příslušná bitmapa zapíše vždy za IFD. Tato struktura je nejpoužívanější. V posledním případě je vidět, že bitmapová data byla zapsána jako první a až za nimi byly zapsány IFD. Toto zdánlivě nesmyslné uspořádání vznikne tehdy, pokud jsou bitmapová data k dispozici pro zapsání ještě před informacemi v IFD. [3]



Obr.31 Tři typy uspořádání dat formátu TIFF [3]

Obr.32 ukazuje jakým způsobem jsou spolu spojeny datové struktury formátu TIFF [3].



Obr.32 Logická organizace souboru TIFF [3]

Formát TIFF podporuje barevný prostor RGB, CMYK,  $YCbCr$  a  $CIE L^*a^*b^*$  [38].

CMYK může mít hodnotu tagu  $SamplesPerPixel = N$  (počet inkoustů - např.  $N = 4$  pro CMYK, protože jsou zde položky modrozelená, fialová, žlutá, a černá),  $BitsPerSample = 8 \times SamplesPerPixel$  (pro tento případ - 8, 8, 8, 8) a  $PhotometricInterpretation = 5$ . Pro  $BitsPerSample$  jsou podporovány pouze osmibitové položky. [38]

$YCbCr$  musí mít mimo jiné tag  $SamplesPerPixel$  nastaven na hodnotu 3 (položky Y,  $C_b$  a  $C_r$ ), tag  $BitsPerSample$  na hodnotu 8, 8, 8 (zápis ve dvou Bytech) a zvolenou kompresi LZW, JPEG a nebo žádnou. [38]

Soubory TIFF obsahují pouze bitmapová data. Nebyl by ovšem problém přidat pár tagů, které by umožnily pracovat s vektorem či textem. Předlohy TIFF 6.0 jsou většinou uloženy v dlaždicích a ne v pruzích. [3]

Pruh je sada jednoho nebo více za sebou jdoucích řádků bitmapových dat předlohy. Tím, že se data rozdělí na pruhy, stává se bufferování, náhodný přístup a mezi ukládání dat mnohem jednodušším. Tento pojem existuje i v jiných formátech, kde se nazývají bloky, stupy nebo shluky. Pruhy ve formátu TIFF se od nich liší v několika důležitých věcech, které vyplývají ze struktury formátu TIFF. [3]

Pro definici pruhů bitmapových dat v souboru TIFF jsou potřeba tři tagy. Jsou to RowsPerStrip (počet řádků v pruhu), StripOffset (pole offsetových hodnot - pozice prvního Bytu každého pruhu) a StripByteCounts (velikost jednoho pruhu v Bytech). [3]

Ke zjištění počtu pruhů se použije tag RowsPerStrip, tag ImageLength a následující výpočet: [3]

$$\text{PočetPruhů} = \text{floor}\left(\frac{\text{ImageLength} \cdot (\text{RowsPerStrip} - 1)}{\text{RowsPerStrip}}\right)$$

TIFF 6.0 zavedl bitmapová data rozdělená do dlaždic, a ne do pruhů. Pruh je jednorozměrný objekt, který má pouze délku. Naproti tomu dlaždice může být brána jako dvourozměrný pruh, který má výšku a šířku, tedy parametry podobné bitmapě. Na každou dlaždici v bitmapě lze nahlížet jako na malou bitmapu obsahující část větší bitmapy. Ukládání komprimovaných dat do dlaždic také v jistém smyslu optimalizuje dekompresi. [3]

Při použití dlaždic místo pruhů se tři tagy - RowsPerStrip, ByteCounts a StripOffsets - zamění za tagy TileWidth (výška dlaždice), TileLength (šířka dlaždice), TileOffsets (pole offsetů na první Byte každé dlaždice) a TileByteCounts (počet Bytů v každé komprimované dlaždici). Tagy pro dlaždice se používají úplně stejným způsobem jako tagy pro pruhy. Předlohy jsou rozděleny buď do pruhů nebo do dlaždic - nikdy ne kombinovaně. [3]

Při dlaždicovém rozdělení malých předloh se nedosahuje příliš výrazných kompresních poměrů. Rovněž používání dlaždic, které jsou velké a vyžadují přidání velkého množství vycpávky (doplňujících Bytů) je nevhodné. [3]

Hodnoty tagu TileWidth a TileLength mohou být použity také k tomu, aby určily počet dlaždic v podsouborech předloh: [3]

$$\text{TilesAcross} = \frac{\text{ImageWidth} + \text{TileWidth} - 1}{\text{TileWidth}}$$

$$\text{TilesDown} = \frac{\text{ImageLength} + \text{TileLength} - 1}{\text{TileLength}}$$

$$\text{TilesInImage} = \text{TilesAcross} \cdot \text{TilesDown} \cdot \text{SamplesPerPixel}$$

## 4.7 JPEG 2000

Vznik standardu JPEG 2000 byl výsledkem snahy najít kompresní metodu vhodnou pro fotografie, která by efektivněji redukovala požadavky na prostor pro uložení obrazu, a také umožňovala pohodlnou práci s tímto obrazem - editaci, extrakci částí obrazu, získání náhledů v různých rozlišeních apod. Maximální velikost předlohy je 4Gb×4Gb pixelů. [39]

Formát JPEG 2000 nabízí široké možnosti pro progresivní přenos s ohledem na různá kritéria - z dat, která postupně přicházejí po přenosové lince, je možné rekonstruovat obraz v nízké kvalitě a s přibývajícím daty kvalitu zvyšovat. Přístup k částem obrazu je tedy možný bez nutnosti dekomprese celého obrazu. [39]

Ve formátu jsou podporovány operace s obrazem (rotace (90°, 180°, 270°), zrcadlení) v komprimovaném stavu bez nutnosti dekomprese, dále také různé barevné modely s různým počtem složek. Je zde také vysoká odolnost proti chybám v bitovém streamu, možnost využití libovolných metadat, možnost zpracování složených dokumentů (zejména text a grafiku), možnost pracovat s počítačovou grafikou s ostrými přechody atd. [39]

Ve formátu JPEG 2000 existují dva typy hlaviček - hlavní hlavička (na začátku kódového streamu) a hlavička dlaždic (tile-part) (na začátku každé dlaždice). JPEG 2000 používá tzv. segmenty značek (marker). Nastavení značek a jejich segmentů ovlivňuje vlastnosti souboru JPEG 2000. Hlavičky jsou v podstatě kolekce značek a segmentů. [40]

Hlavní hlavička vyžaduje minimálně 4 segmenty značek - viz *Tab.26*. Hlavička dlaždic vyžaduje minimálně 2 segmenty značek - viz *Tab.27*. [40]

*Tab.26 Povinné segmenty v hlavní hlavičce formátu JPEG 2000 [40]*

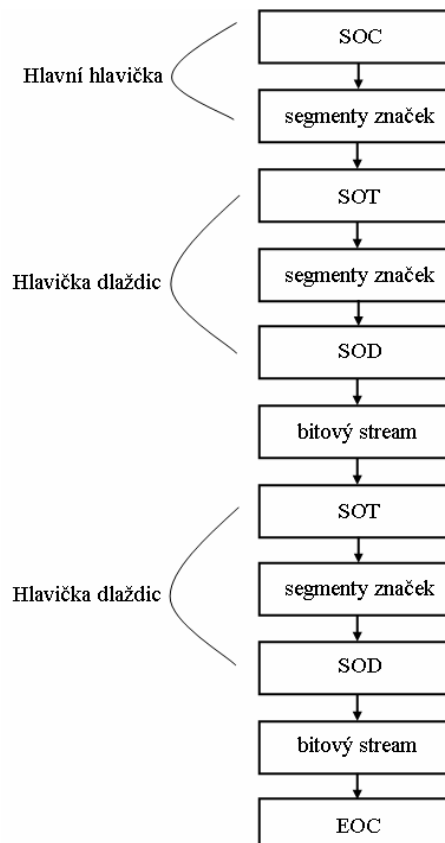
Segment	Kód	Význam
SOC (start of codestream)	FFh 4Fh	označuje počátek kódového streamu - vždy na prvním místě
SIZ (Image and tile size)	FFh 51h	informace o rozměrech obrázku a jedné dlaždice - vždy na druhém místě
COD (Coding style default)	FFh 52h	informace o typu kódování
QCD (Quantization default)	FFh 5Ch	informace o kvantizaci

Tab.27 Povinné segmenty v hlavičce dlaždic formátu JPEG 2000 [40]

Segment	Kód	Význam
SOT (Start of tile part)	FFh 90h	označuje počátek dlaždice - vždy první
SOD (Start of Data)	FFh 93h	označuje počátek dat

Segment značky zahrnuje značku a parametry. Každá značka má 2 Byty - první Byte má vždy hodnotu FFh, druhý Byte obsahuje hodnotu v rozmezí 01h až FEh. Za těmito Byty by měly být bezznaménkové celočíselné hodnoty (unsigned integer) v uspořádání velký endian, které symbolizují délku parametrů v Bytech (včetně tohoto údaje o délce). Tento údaj se nevyskytuje pouze u segmentů značek SOC, SOD, EOC (End of codestream) a EPH (End of packet header), které obsahují jen kódovou hodnotu. [40]

Obr.33 zobrazuje příklad konstrukce kódového streamu. [40]

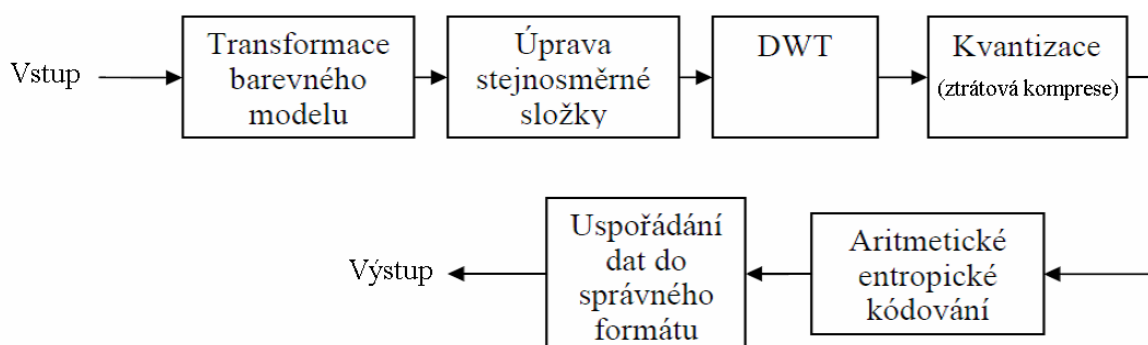


Obr.33 Kódový stream formátu JPEG 2000 [40]

Kromě ztrátové komprese nabízí JPEG 2000 rovněž kompresi bezztrátovou. Pro kompresi se využívá Diskrétní vlnková (wavelet) transformace (DWT). Bezztrátová varianta běžně dosahuje kompresního poměru 1:2, tedy data po kompresi mají poloviční velikost než před kompresí. [39]

Obraz je před DWT rozdělen do stejně velkých (oblasti u okrajů obrazu se mohou velikostně lišit) obdélníkových oblastí (tzv. tile - dlaždice). Velikost těchto oblastí může uživatel nastavit až na velikost celého obrazu. Obraz je pak zpracován DWT najednou. Rozdělení na části snižuje paměťové nároky, ale snižuje také kvalitu obrazu (mohou vznikat viditelné hranice mezi oblastmi). [39]

Řetězec operací, které jsou prováděny během komprese JPEG 2000 je naznačen na *Obr.34*. Při dekompresi jsou použity inverzní transformace v přesně opačném pořadí. [39]



*Obr.34 Postup komprese formátu JPEG 2000 [39]*

Prvním krokem je transformace barevného modelu. Nejčastěji je obraz uložen v modelu RGB. JPEG 2000 může pracovat i s jinými modely, které mají např. více než tři složky - pak se tento krok neprovádí. Základní nevýhodou modelu RGB při kompresi je, že mezi složkami je velká míra korelace. Jinými slovy, velká část informace se opakuje ve všech barevných složkách. Pro kompresi se model transformuje na YUV, který má míru korelace nižší. Složky YUV se vypočtou jako lineární kombinace složek RGB. Pokud se provádí bezztrátová komprese, použije se celočíselná aproximace těchto vztahů, kde není třeba zaokrouhlovat a nevzniká tak chyba. [39]

Úprava stejnosměrné složky je velmi jednoduchá operace, která v podstatě znamená přechod od neznaménkového vyjádření složek na znaménkové vyjádření. [39]

Následuje diskrétní vlnková transformace - pro ztrátovou transformaci se používá Cohen-Daubechies-Feauveau 9/7, pro bezztrátovou kompresi se používá celočíselná varianta Cohen-Daubechies-Feauveau 5/3. Účelem této transformace je reprezentovat původní obraz tak, aby následující části komprese byly schopny co nejefektivněji odstranit redundantní informace. Transformace spočívá v použití jednoduchých filtrů, které vstupní posloupnost rozloží na dvě pásma - pásmo nízkých kmitočtů a pásmo vysokých kmitočtů. V JPEG 2000 se provádí dvojrozměrná DWT (2D DWT). Pro výpočet využijeme separability 2D DWT. Tato vlastnost umožňuje vypočítat 2D DWT ve dvou krocích pomocí jednorozměrné DWT. Nejprve se aplikuje jednorozměrná DWT na všechny řádky a pak na všechny sloupce výsledku předešlého kroku. Vzniknou tak čtyři pásma - LL (nízké kmitočty horizontálně i vertikálně), LH (nízké kmitočty horizontálně, vysoké vertikálně), HL (vysoké kmitočty horizontálně, nízké vertikálně), HH (vysoké kmitočty horizontálně, vysoké vertikálně). Na pásmo LL se může použít 2D DWT opakovaně a dostat tak další úroveň dekompozice. Úroveň dekompozice také ovlivňuje kvalitu komprese a je nastavitelná uživatelem. [39]

Při ztrátové kompresi, se provede tzv. kvantizace. V podstatě jde o snížení přesnosti koeficientů DWT. Provádí se vydělením koeficientů kvantizačním krokem a zaokrouhlením. V tomto kroku dochází k nevratné redukci informace v obraze. Se zvyšující se kvantizací se vytrácí detaily, obraz se jakoby rozmazává, ale prostorové návaznosti zůstávají zachovány a nevznikají žádné ostré přechody. Obraz tak vypadá méně narušený a přirozenější. [39]

Následuje nejsložitější část komprese - aritmetické entropické kódování. Metoda, která je použita v JPEG 2000, se nazývá EBCOT (Embedded Block Coding with Optimal Truncation). Do tohoto bloku vstupují bloky koeficientů DWT (např. 64×64 koeficientů pro jeden blok) a výstupem je komprimovaný bitový stream. EBCOT je poměrně složitý adaptivní proces, který je pro kompresi velmi účinný, ovšem také výpočetně náročný. Komprese, která v něm probíhá, je bezztrátová. Bitový stream má však tu vlastnost, že je možné jej zkrátit, což má podobný efekt jako větší kvantizace provedená v předešlém kroku. [39]

Poslední blok je zodpovědný za to, aby data byla uspořádána ve správném pořadí a byly k nim přidány správné hlavičky [39].

## 4.8 HD Photo

Tento formát je známý také pod názvem Windows Media Photo. HD Photo bylo navrženo k dosažení co nejvyšší obrazové kvality a optimálního výkonu. Soubor může obsahovat více obrázků a metadata. Maximální velikost souboru je  $2^{32}-1$  Bytů. [41]

HD Photo nabízí kódování s vysokým obrazovým rozsahem (HDR) a podporuje širokou škálu barevné reprezentace pixelů - monochromatickou, RGB, CMYK nebo n-kanálovou. Barevné složky lze popsat 8 nebo 16-bitovými bezznaménkovými celočíselnými hodnotami, 16 nebo 32-bitovými znaménkovými celočíselnými hodnotami a 16 nebo 32-bitovými hodnotami s plovoucí desetinnou čárkou. 8 a 16-bitové hodnoty jsou podporovány jak v neztrátové, tak ztrátové kompresi, ale 32-bitové mají podporu jen pro ztrátovou kompresi. Možné kombinace pixelových vyjádření pro barevné prostory RGB, RGBA a obrazy v odstínech šedi jsou uvedeny jako Příloha P I. [41]

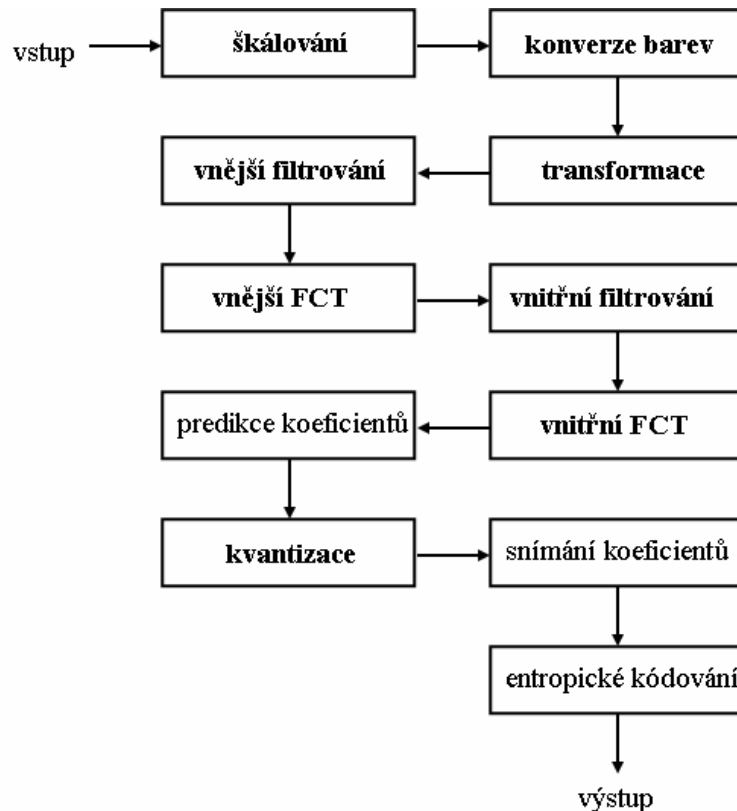
Struktura formátu HD Photo je téměř totožná se strukturou formátu TIFF. Hlavička souboru má délku 8 Bytů, po ní následují adresáře souboru předlohy (Image File Directory - IFD) a poté data. [41]

První 4 Byty mají vždy hodnotu 49h 49h, což značí, že jsou hodnoty vždy ukládány v uspořádání malý endian. Dále je uložen identifikátor formátu HD Photo - BCh. Nakonec obsahuje hlavička verzi formátu - buď hodnota 00h (0. verze) nebo 01h (1. verze). Poslední 4 Byty udávají offset prvního IFD. IFD je stejně jako ve formátu TIFF složen z tagů. V HD Photo jsou přidány nové typy tagů, ale je zde také podpora tagů formátu TIFF. [41]

HD Photo využívá kompresní algoritmus optimalizovaný pro digitální fotografie. Ve formátu je implementována neztrátová i ztrátová komprese. Neztrátová komprese dokáže podle autorů zmenšit velikost obrázku 2,5-krát jeho původní velikosti bez komprese. [41]

HD Photo používá reverzibilní barevnou konverzi, reverzibilní biorthogonální transformaci (druh DWT) a pokročilé kódovací schéma nearitmetické entropie. Kompresní algoritmus je navrhnut pro rychlou kompresi a dekompresi. Jádro komprese vyžaduje nanejvýš 3 netriviální (násobení a přičítání) a 7 triviálních (přičítání nebo bitový posun) operací na jeden pixel. Nevyskytuje se zde tedy žádná operace dělení. V módu zaměřeném na nejvyšší výkon jsou vyžadovány pouze 1 netriviální a 4 triviální operace na pixel. Obrázek je rozdělen na  $16 \times 16$  makro bloky, což poskytuje minimální paměťové požadavky. [41]

Na *Obr.35* je znázorněn postup komprese formátu HD Photo. Škálování je prováděno na vstupní data, která jsou delší než 24 bitů. Transformace znamená rozdělení obrazu na bloky a makrobloky. FCT znamená hlavní dopřednou transformaci (Forward Core Transformation) a je jádrem celého kompresního schématu. Kvantizace se provádí jen u ztrátové komprese. [42]



*Obr.35 Postup komprese formátu HD Photo*

## 4.9 JPEG-LS

Standard JPEG - Lossless (JPEG-LS) je také znám pod názvem LOCO-I (Low Complexity LOSSless COmpression for Images). Formát využívá neztrátovou a mírně ztrátovou (near lossless) kompresi. [43]

Příklad struktury formátu JPEG-LS je vidět v *Tab.28*. Tato struktura se však může částečně měnit, např. při obsazení více složek ve snímku (složkou může být myšlena např. barva R barevného prostoru RGB) nebo více skenovacích segmentů. [44]

Tab.28 Základní struktura formátu JPEG-LS [44]

Offset	Velikost [Byte]	Význam
0	2	počátek obrázku - vždy FFh D8h
2	2	počátek snímku - vždy FFh F7h
4	2	délka segmentu
6	1	počet bitů na vzorek
7	2	počet řádků
9	2	počet sloupců
11	1	počet složek ve snímku
12	1	ID složky
13	1	informace o podvzorkování složky
14	1	vždy 0
15	2	počátek skenovacího segmentu - vždy FFh DAh
17	2	délka segmentu
19	1	počet složek v tomto skenovacím segmentu
20	1	ID složky ve skenovacím segmentu
21	1	index mapovací tabulky (0-bez mapovací tabulky)
22	1	tolerance při predikci (0-pro neztrátovou kompresi)
23	1	prokládaný mód (0-bez prokládání)
24	1	transformace bodu (0-bez transformace)
25	N	zkomprimovaná data obrázku
25+N	2	konec obrázku - vždy FFh D9h

Při kompresi formátu JPEG-LS se prohledává několik z předchozích sousedů aktuálního pixelu a zjišťuje se spojitost (resp. rozdíly) mezi pixely. Tyto spojitosti se používají k predikci pixelu a následně k výběru pravděpodobnostní distribuce. [8] Fixní prediktor zjišťuje jednoduchým testem horizontální a vertikální hrany, algoritmus je následující: [43]

```

1 if (c >= max(a,b))
2   xMED = min(a, b)
3 elseif (c <= min(a,b))
4   xMED = max(a, b)
5 else
6   xMED = a + b - c

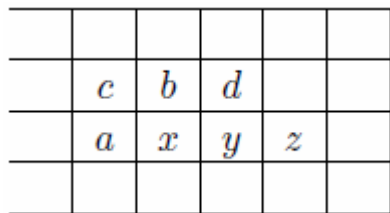
```

Následně se aplikuje adaptivní korekce chyby, způsobené tímto fixním prediktorem [44].

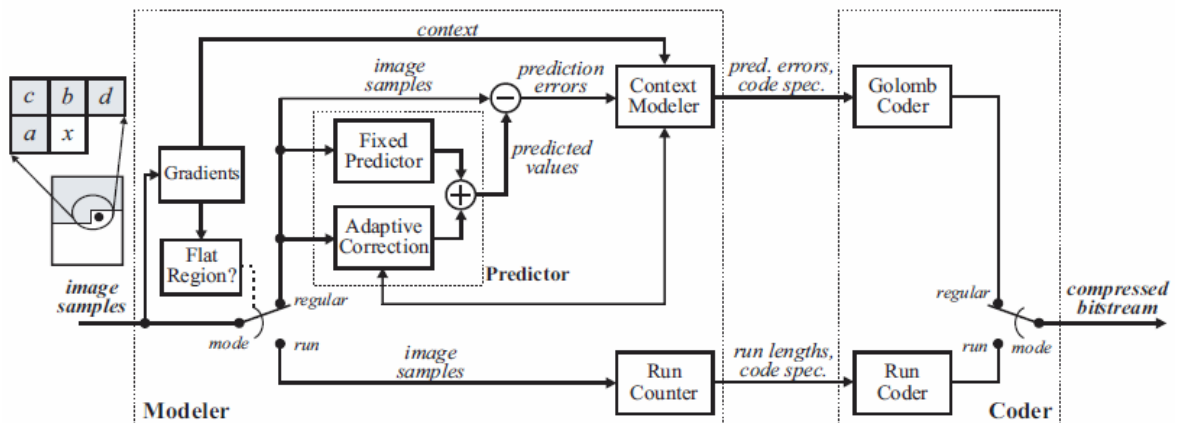
Pravděpodobnostní distribuce se využívá ke kódování chyby predikce pomocí Golombova kódování. [8] Golombovo kódování je speciální případ Huffmanova kódování

přizpůsobený geometrické distribuci [44]. Je zde také možnost využití módu Run (RLE) [8].

Na *Obr.36* je vidět aktuální pixel  $x$ , na který je predikce aplikována. Kodér prozkoumá spojitosti mezi pixely a rozhodne, zda kódovat pixel  $x$  v módu Run nebo v módu Regular. Pokud z kontextu vyplývá, že následující pixely  $y$ ,  $z$ , atd. jsou pravděpodobně stejné, kodér zvolí mód Run. Jinak je použit mód Regular. Na *Obr.37* je možné spatřit postup komprese. [8] [43]



*Obr.36* Pozice sousedních hodnot využívaných predikcí ve formátu JPEG-LS [8]



*Obr.37* Postup komprese formátu JPEG-LS [43]

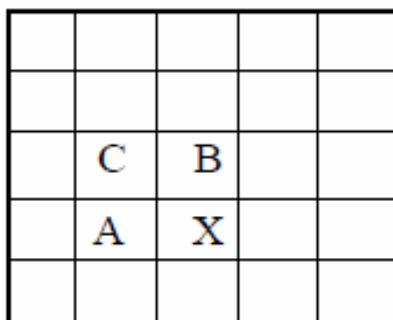
JPEG-LS využívá téměř symetrickou kompresní metodu. Zkomprimovaný stream obsahuje segment s daty (s Golombovými kódy a zakódovanými proudy dat pomocí RLE), segmenty značek (s informacemi potřebnými pro dekodér) a značky. Značka znamená Byte následovaný speciálním kódem, signalizující počátek nového segmentu značky. [8]

### 4.9.1 Lossless JPEG

Formátu JPEG-LS předcházela tzv. Lossless JPEG (LJPEG), který je ovšem založen na jiném principu a není tedy možné zaměňovat tyto dva formáty [45].

LJPEG pracuje s předlohami s 2 až 16 bity na vzorek. Vzorek znamená jeden element v dvourozměrném poli (např. jednu hodnotu složky R barevného prostoru RGB). Nejlepších výsledků dosahuje při šesti bitech a více. Pro tyto předlohy se dosahuje kompresních poměrů okolo 1:2. Při menším počtu bitů na pixel je lepší použít jinou kompresní metodu. LJPEG nebyl příliš populární z důvodu, že formát PNG, má u většiny obrazů lepší kompresi. Obecně kvalitnější kompresi vykazuje také JPEG-LS. [3] [44] [45] [46]

LJPEG používá dvourozměrné diferenční pulzně kódované schéma. Základní předpoklad je ten, že hodnota pixelu je kombinována s hodnotami až tří sousedních pixelů tak, aby byla vytvořena predikovaná hodnota. Tato hodnota je poté odečtena od původní hodnoty pixelu. Na *Obr.38* je vidět hodnota hledané predikce X a sousední hodnoty A, B, C. V *Tab.29* jsou zobrazeny možné typy predikce pro X. Typy 1-3 jsou označovány jako jednorozměrné predikce, typy 4-7 jako predikce dvourozměrné. [3] [46]

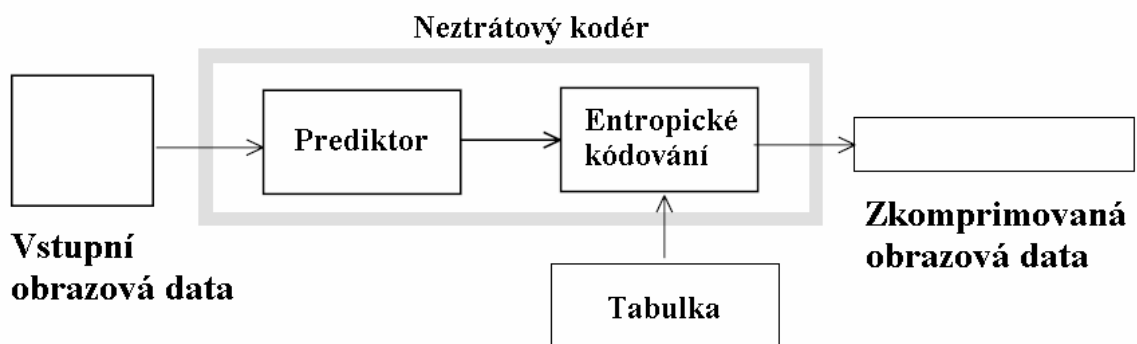


*Obr.38* Pozice sousedních hodnot využívaných predikcí ve formátu LJPEG [46]

Tab.29 Typy predikce  
formátu JPEG [46]

Hodnota	Typ predikce
0	žádná predikce
1	A
2	B
3	C
4	A+B-C
5	$A+(B-C)/2$
6	$B+(A-C)/2$
7	$(A+B)/2$

Až je takto zpracována celá bitmapa, jsou výsledné predikce komprimovány buď Huffmanovou nebo binárně aritmetickou entropní kódovací metodou. Schéma komprese JPEG je vidět na Obr.39. [3] [46]



Obr.39 Postup komprese formátu JPEG [46]

## 4.10 OpenEXR

OpenEXR je open-source formát pro počítačovou grafiku, patřící do kategorie High Dynamic Range (HDR) - obrazy s vysokým dynamickým rozsahem [47]. Jedná se o formát, který ukládá jeden pixel do 16-bitové nebo 32-bitové hodnoty s plovoucí desetinnou čárkou a nebo 32-bitové hodnoty bez znaménka [48].

Formát nabízí výrazně vyšší dynamický rozsah, než stávající 8- a 10-bitové formáty. Podle vyjádření tvůrců je dynamický rozsah formátu zhruba 30 f-stops (expozičních čísel) bez

ztráty věrnosti a dalších 10 f-stops navíc při minimální ztrátě - stávající 8-bitové formáty poskytují dynamický rozsah okolo 7-10 f-stops. Při 16-bitovém vyjádření je zároveň dosaženo podstatně vyššího barevného rozsahu - v rámci OpenEXR je to 1024 barevných hladin na 1 f-stop oproti 20 až 70 hladin současných 8-bitových formátů. [47] [48]

Spolu s formátem byly vyvinuty odpovídající programové interface, napsané v jazycích C++ a C. Při návrhu bylo použito pouze standardních funkcí těchto jazyků (podle normy ISO), což zaručuje nezávislost OpenEXR na platformě (hardware, operační systém). [47]

Na začátku souboru OpenEXR jsou vždy hodnoty 76h 2Fh 31h 01h. Dále následuje hodnota verze, která je datového typu int. Verze může aktuálně nabývat dvou hodnot - 202h (pro dlaždicové uložení pixelů) a 02h (pro uložení pixelů v pružích). Hodnoty se ukládají v uspořádání malý endian. [49]

Po identifikačním čísle a čísle verze je uložena hlavička. Hlavička je sekvence atributů oddělených nulovým Bytem. Atribut obsahuje jméno, datový typ, velikost a hodnotu. Každá hlavička formátu OpenEXR musí obsahovat minimálně atributy, které jsou obsaženy v *Tab.30*. [49]

*Tab.30 Povinné atributy v hlavičce formátu OpenEXR [48] [49]*

<b>Atribut</b>	<b>Typ</b>	<b>Význam</b>
channels	chlist	popis barevných kanálů
compression	compression	aplikovaná kompresní metoda
dataWindow	box2i	informace o datovém okně
displayWindow	box2i	informace o zobrazovacím okně
lineOrder	lineOrder	specifikuje seřazení pruhů nebo dlaždic
pixelAspectRatio	float	šířka / výška pixelů
screenWindowCenter	v2f	popisuje perspektivu - pro 2D zobrazení má hodnotu (0, 0)
screenWindowWidth	float	popisuje perspektivu - pro 2D zobrazení má hodnotu 1
tileDescription	tiledesc	popisuje dlaždice - atribut je nutný jen pro obrázky s dlaždicemi

Po hlavičce je uložena tabulka offsetů, která umožňuje náhodný přístup k jednotlivým pruhům nebo dlaždicím. Poté následují samotné pruhy (scan lines) nebo dlaždice (tiles). [49]

Pruhy jsou ukládány po blocích. To kolik pruhů bude uloženo v jednom bloku závisí na použité kompresi. Obrázky uložené pomocí dlaždic umožňují náhodný přístup k obdélníkovému sub-regionu obrázku. V jednom souboru OpenEXR mohou být uloženy dlaždicové obrazy s jiným rozlišením. Tohoto se obecně využívá pro textury 3D objektů (kvůli zrychlení filtrování textur) a také pro rychlé zoomování v programech, které zobrazují velmi velké obrázky. [48] [49]

Formát OpenEXR v sobě dokáže nést i doplňkové informace, související např. se způsobem jejich vytvoření (pozice kamery, směr pohledu apod.). Tyto informace lze zpracovat odpovídajícím způsobem v příslušném software, a naopak bez problému ignorovat v aplikacích, které z OpenEXR formátu pouze potřebují načíst obrazové informace. [47]

Každý obrázek OpenEXR obsahuje jednu nebo více obrazových kanálů. Každý kanál má jméno, datový typ a hodnoty s pozicí x, y. Jméno kanálu je uloženo v textovém řetězci - např. „R“, „Z“ nebo „yVelocity“. Jméno říká programu, jak má interpretovat data v barevném kanálu. Obrázky OpenEXR mohou obsahovat libovolné hodnoty a kombinace obrazových kanálů, např. červenou, zelenou, modrou a alfu; světlost a chroma kanály; hloubku, povrch, nebo pohybové vektory. [48]

Ve formátu OpenEXR je definováno tzv. datové okno a zobrazovací okno (display window). Obě okna mají tvar obdélníku a jejich velikost definují souřadnice x, y počátku a x, y konce okna. Datové okno obsahuje všechny pixely. Zobrazovací udává oblast, kterou je možné zobrazit - v tomto okně nemusí být umístěny všechny pixely a také může pokrývat větší oblast než datové okno. Názorné ukázky datového a zobrazovacího okna jsou uvedeny jako Příloha P II. [48]

Většina metod datové komprese implementovaných v OpenEXR je neztrátových. OpenEXR také podporuje ztrátovou kompresi a možnost uložení dat bez komprese. Podle tvůrců budou v budoucnu implementovány další typy neztrátové i ztrátové komprese. Ve formátu OpenEXR jsou implementovány kompresní metody, které znázorňuje *Tab.31*. [48] [49]

Tab.31 Kompresní schémata formátu OpenEXR [48] [49]

Kompresce	Neztrátová	Počet pruhů na blok
PIZ	ano	32
ZIP	ano	1 nebo 16
RLE	ano	1
PXR24	ne	16
B44	ne	32
B44A	ne	32

Kompresce PIZ se provádí aplikací vlnkové (wavelet) transformace na data pixelů a výsledek je poté zakódován Huffmanovým kódováním. Kompresce PIZ pracuje dobře jak s obrázkovými daty uloženými v pruzích, tak s daty uloženými ve velkých dlaždicích. Soubory jsou komprimovány a dekomprimovány přibližně stejnou dobu. Použitím PIZ komprese na fotografie s významným množstvím změn se velikost souboru zmenší o 35 až 55 % velikosti souboru bez komprese. Tato komprese je uzpůsobena obrázkům, které vytváří firma Industrial Light & Magic. [48]

Kompresce ZIP je provedena pomocí open source knihovny zlib. Doba dekomprese je u ZIP kratší než u PIZ, ale doba komprese je u ZIP významně pomalejší. Vzhledem k tomu, že pro textury je lepší menší doba dekomprese a nezáleží příliš na zmenšení souboru, je pro tento účel vhodnější použít metodu ZIP. Použitím ZIP na fotografie dojde ke zmenšení souboru o 45 až 55 % velikosti souboru bez komprese. [48]

Kompresce RLE je rychlá a nejlépe funguje při použití na velké konstantní oblasti. Pro fotografie se obvykle pohybuje zmenšení souboru mezi 60 a 75 % velikosti souboru bez komprese. [48]

## 4.11 WebPll

Nová neztrátová komprese formátu WebP (WebPll) je v této době na bázi experimentu. Formát WebPll je pouze dočasný a později splyne s formátem WebP. Kromě vydaných utilit neexistuje žádný program, který tento formát využívá a vzhledem k budoucímu sloučení s formátem WebP se takový nápad jeví jako nevhodný. Formát používá barevný prostor RGBA. [50] [51]

Neztrátová komprese je postavena na transformování obrazu použitím několika rozdílných technik a následného použití entropického kódování na transformované parametry a

transformované obrazové data. Transformace aplikovaná na obraz obsahuje prostorovou predikci pixelů, transformaci barevného prostoru, používá lokální palety, zkomprimuje více pixelů do jednoho a nahradí alfu. Pro entropické kódování je zde použita varianta komprese LZ77 a Huffmanovo kódování, které používá 2D hodnoty kódové vzdálenosti. [51]

Při testování balíku obrázků o velikosti 18625893 Bytů došlo k celkové redukci o 28,2 % v porovnání s formátem PNG. 99,4 % obrázků bylo zkomprimováno lépe použitím formátu WebPll než formátu PNG. [51]

Kompresní a dekompresní utility zatím nebyly optimalizovány pro rychlost [51]. Podle [52] došlo při testování po sedmi hodinách provádění komprese na jednom souboru k havárii programu s hlášením o nedostatku paměti. Podobné informace o dlouhé době komprese a nestabilitě lze nalézt také zde: [53]. Větší problém než nestabilita je velká doba komprese, která běžně dosahuje řádů desítek minut až několika hodin. Velikost souboru na použité kvalitě neztrátové komprese příliš nezávisí - rozdíl je v řádu stovek, u větších souborů tisíců kB - ale výrazně ovlivňuje dobu komprese (někdy i v řádu minut). Doba dekomprese je rychlá. [53] [54]

Podle [53] je na nekomprimované soubory o velikosti větší než 100 MB nejlepší použít kvalitu 0, na soubory větší než 30 MB kvalitu 10 - jen nepatrné zlepšení komprese lze zaznamenat u použití kvality 30.

## **II. PRAKTICKÁ ČÁST**

## 5 APLIKACE PRO TESTOVÁNÍ KOMPRESNÍCH ALGORITMŮ

Aplikace má název: „Grafický prohlížeč“. Byla vyvíjena pod operačním systémem Windows 7 Professional a ovládá se pomocí myši. Aplikace je naprogramována v jazyce C++ pomocí vývojového prostředí CodeLite, překladače MinGW a knihovny wxWidgets [55]. GUI je navrženo a generováno v aplikaci wxFormBuilder [56].

K funkčnosti aplikace jsou kromě hlavního spustitelného souboru GrafickyProhlizec.exe zapotřebí také soubory, jejichž seznam je vypsán v *Tab.32* („console\“ znamená název složky, ve které musí být uloženy dané konzolové aplikace).

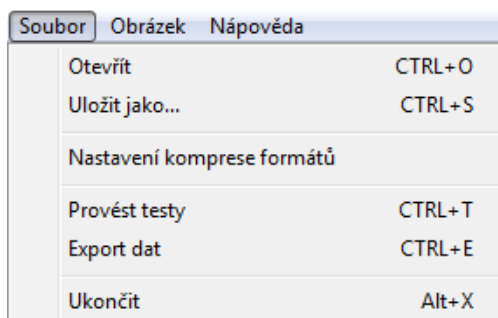
*Tab.32 Seznam souborů nutných pro spuštění aplikace*

Název	Použití	Zdroj
mingwm10.dll	knihovna MinGW (základ aplikace)	[55]
wxmsw28u_gcc_custom.dll	multiplatformní GUI knihovna wxWidgets 2.8.12 (základ aplikace)	[55]
FreeImage.dll	Knihovna pro práci s obrázky běžných formátů (BMP, PNG, TIFF atd.)	[57]
console\JBIGtoPNM.exe console\PNMtoJBIG.exe	konzolové aplikace pro dekódování a kódování formátu JBIG	[58]
console\WMPDecApp.exe console\WMPEncApp.exe	konzolové aplikace pro dekódování a kódování formátu HD Photo	[59]
console\locod.exe console\locoe.exe	konzolové aplikace pro dekódování a kódování formátu JPEG-LS	[60]

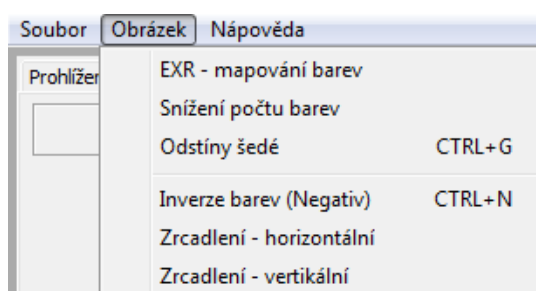
Vzhledem k tomu, že knihovna FreeImage ani knihovna wxWidgets plně nepodporuje formáty PCX, je v aplikaci využito vlastní řešení načítání a ukládání tohoto formátu. Výjimkou je ukládání 24 bitových obrázků PCX, které je provedeno s využitím wxWidgets.

### 5.1 Možnosti aplikace

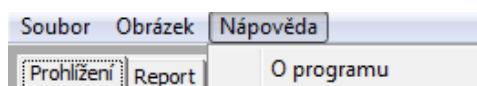
Aplikace má tři hlavní záložky v menu - Soubor ; Obrázek ; Nápověda. Obsah těchto záložek je vidět na *Obr.40*, *Obr.41* a *Obr.42*.



Obr.40 Záložka v menu - Soubor



Obr.41 Záložka v menu - Obrázek



Obr.42 Záložka v menu - Nápověda

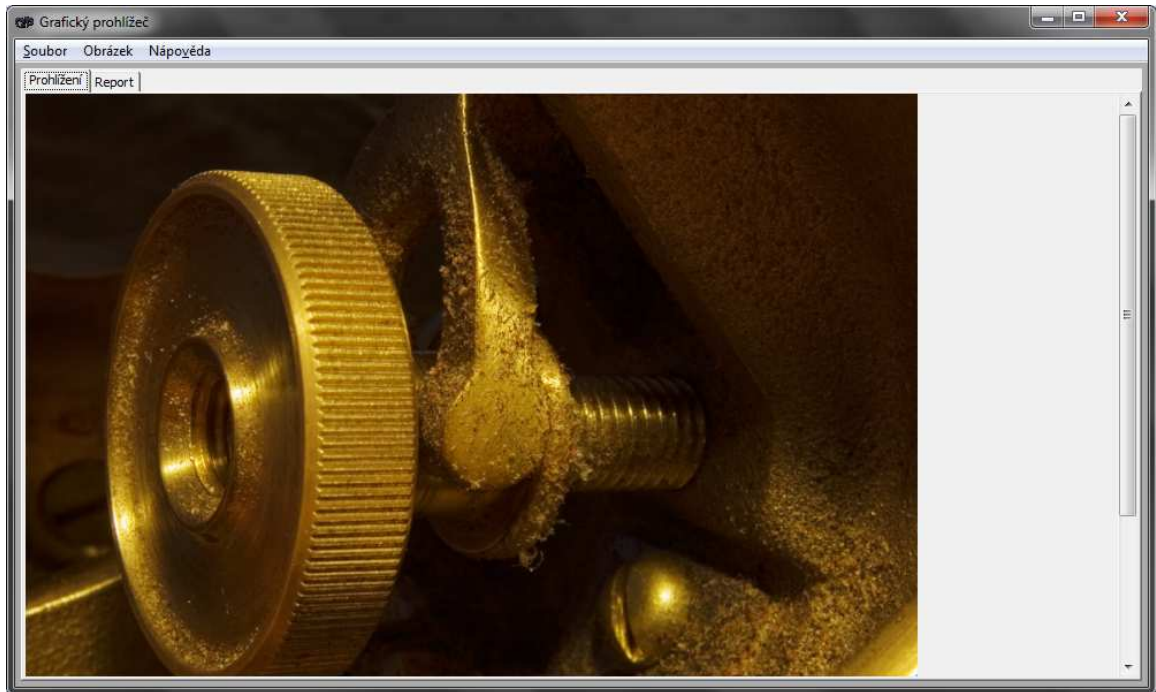
### 5.1.1 Menu Soubor

Pro otevření souboru s obrazovými daty slouží položka „Otevřít“, pro jeho uložení do příslušného formátu lze použít položku „Uložit jako...“.

Aplikace podporuje formáty BMP (i s kompresí RLE), GIF, HD Photo (1, 8 (odstíny šedi), 24, 32 bitů, Gray FLOAT, RGBAF (HD Photo nepodporuje RGBF, ale jen RGBAF s možností ignorovat alfa kanál)), JBIG (1, 8 (odstíny šedi) bitů), JPEG 2000 (8 (odstíny šedi) 24 a 32 bitů), JPEG-LS (8 (odstíny šedi) 24 bitů), OpenEXR (Gray FLOAT, RGBF), PCX, PNG, TGA (>8 bitů), TIFF (podporuje i Gray FLOAT a RGBF).

Díky knihovně FreeImage lze otevřít i formáty jako JPEG, PBM, PGM, PPM, formáty fotoaparátů (RAW) jako BMQ, DNG, NEF a další (kompletní soupis formátů je možné

nalézt v dokumentaci knihovny FreeImage [61]). Prioritně však nejsou aplikací podporovány, a tak lze tyto soubory otevřít jen zápisem jejich názvu v dialogu pro otevření. V záložce prohlížení je možné zhlédnout aktuálně načtený obrázek (viz *Obr.43*).



*Obr.43 Prohlížení obrázků v aplikaci*

Při otevření/uložení souboru se měří doba, za kterou byl soubor otevřen/uložen. Při ukládání se dále vypočte kompresní poměr a veličina, která byla nazvána jako „Časová efektivita komprese“. Tuto veličinu lze popsat následujícím vzorcem:

$$\text{ČEK} = \frac{1 - KP}{DK} \cdot 100 \left[ \frac{\%}{\text{ms}} \right]$$

ČEK - časová efektivita komprese

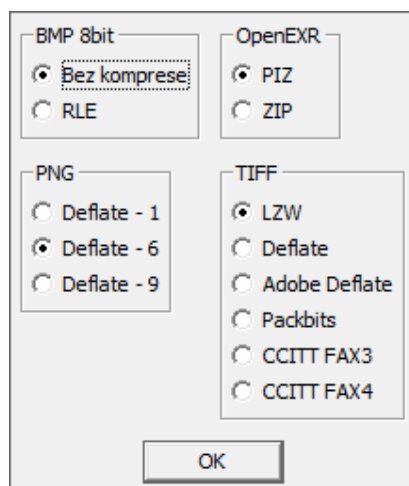
KP - kompresní poměr

DK - doba komprese

Časová efektivita komprese vyjadřuje o kolik procent se zmenší velikost souboru za jednu milisekundu. Při záporné kompresi se tato hodnota pohybuje v záporných číslech - interval možných výsledků je tedy  $(-\infty; 100)$ .

Výše zmíněná veličina je inspirována parametrem popsáním v kapitole 2.2 - konkrétně se jedná o poměr mezi dobou komprese a kompresním poměrem. Tento parametr určuje dobu komprese na jednotku kompresního poměru, přičemž čím vyšší číslo tím lépe, a pokud je výsledek menší než doba komprese došlo k záporné kompresi. Parametr je však na první pohled méně srozumitelný a vyžaduje větší přemýšlení k pochopení jeho účelu, a proto byla použita výše zmíněná modifikace, která vyznívá jasněji.

V položce „Nastavení komprese formátů“ je možné nastavit typ výstupní komprese u formátů BMP, EXR, PNG a TIFF. Konkrétní možnosti nastavení komprese jsou znázorněny na *Obr.44*. U formátu BMP může být RLE komprese využita jen u 8 bitových obrázků. Komprese „Deflate - 1“ označuje nejnižší úroveň komprese Deflate a „Deflate - 9“ naopak úroveň nejvyšší. Při zvolení komprese CCITT FAX3 a FAX4 budou obrázky převedeny na dvoubarevné - převod barev se uskuteční pomocí zvoleného algoritmu v položce „Snížení počtu barev“.



*Obr.44 Dialog s nastavením komprese formátů*

V aplikaci lze automaticky provést testy kompresních algoritmů na větším počtu souborů. Toto se provede pomocí položky „Provést testy“. Před zahájením je potřeba vybrat složku se vstupními obrázkovými daty, přičemž dojde k prohledání i případných podsložek. Dále se zvolí složka, do které se uloží obrázky v různých formátech a popř. i s různými typy kompresí. Obrázky se stejnou příponou a různým typem komprese (jedná se o formáty

uvedené v dialogu „Nastavení komprese formátů“) jsou v názvu souboru doplněny o tvar „\_<typ komprese>“ - např. jedním z možných výstupů obrázku s názvem „obrázek.bmp“ bude „obrázek\_deflate1.png“.

Jako vstupní obrázky v automatickém testu jsou podporovány formáty s příponou BMP, CR2, EXR, GIF, HDR, J2C, J2K, JP2, JPC, JPEG, JPG, NEF, ORF, PFM, PNG, RAF, TGA, TIF a TIFF. Po načtení obrázku se provede až 16 testů kompresních algoritmů - volba algoritmů je na uživateli. Testovat lze formáty a komprese uvedené v *Tab.33*.

*Tab.33 Volitelné formáty a komprese pro provádění testů*

<b>Formát</b>	<b>Komprese</b>
BMP	RLE
EXR	PIZ
EXR	ZIP
GIF	LZW
HD Photo (WDP)	LOSSLESS WAVELET
JBIG	JBIG
JPEG 2000	LOSSLESS WAVELET
JPEG-LS	LOCO-I
PCX	RLE
PNG	DEFLATE 1
PNG	DEFLATE 9
TGA	RLE
TIFF	LZW
TIFF	PACKBITS
TIFF	CCITT FAX 3
TIFF	CCITT FAX 4

Při provádění testů je zobrazen ukazatel průběhu a postupně se do reportu aplikace doplňují parametry jednotlivých souborů (viz *Obr.45*). Průběh testů nelze pozastavit ani zastavit. Doba komprese a dekomprese se měří dvakrát a výsledkem je průměr těchto hodnot.

Název	BH [bpp]	BP	OŠ	Šířka [px]	Výška [px]	DD [ms]	Typ K	DK [ms]	KP	ČEK [% / ms]
08d880fe-a592-483d-8a31-b508716750ea_C_24	24	RGB(A)	N	387	420	93	HDPHOTO - LOSSLESS_WAVE	118	0.222333	0.659040
08d880fe-a592-483d-8a31-b508716750ea_C_24	24	RGB(A)	N	387	420	117	JP2 - LOSSLESS_WAVELET	242	0.142704	0.354254
08d880fe-a592-483d-8a31-b508716750ea_C_24	24	RGB(A)	N	387	420	30	JLS - LOCO-1	30	0.155354	2.815486
08d880fe-a592-483d-8a31-b508716750ea_C_24	24	RGB(A)	N	387	420	16	PCX - RLE	31	0.401864	1.929471
08d880fe-a592-483d-8a31-b508716750ea_C_24	24	RGB(A)	N	387	420	1	PNG - DEFLATE_1	39	0.166280	2.137743
08d880fe-a592-483d-8a31-b508716750ea_C_24	24	RGB(A)	N	387	420	1	TGA - RLE	8	0.340649	8.241884
08d880fe-a592-483d-8a31-b508716750ea_C_24	24	RGB(A)	N	387	420	16	TIFF - LZW	62	0.228505	1.244346
08d880fe-a592-483d-8a31-b508716750ea_C_24	24	RGB(A)	N	387	420	1	TIFF - PACKBITS	78	0.448515	0.707033
101.wdp	24	RGB(A)	N	987	831	515	HDPHOTO - LOSSLESS_WAVE	609	1.230283	-0.037813
101.jp2	24	RGB(A)	N	987	831	804	JP2 - LOSSLESS_WAVELET	1404	0.799196	0.014302
101.jls	24	RGB(A)	N	987	831	116	JLS - LOCO-1	148	0.814968	0.125022
101.pcx	4	RGB(A)	N	301	429	1	PCX - RLE	8	0.499555	6.255565
101_deflate1.png	8	RGB(A)	N	987	831	16	PNG - DEFLATE_1	16	0.144730	5.345441
101.tga	8	RGB(A)	N	987	831	16	TGA - RLE	31	0.449033	1.777312
101_LZW.tiff	8	RGB(A)	N	987	831	47	TIFF - LZW	70	0.346172	0.934040
101_Packbits.tiff	8	RGB(A)	N	987	831	16	TIFF - PACKBITS	78	2.015721	-1.302206
160.wdp	24	RGB(A)	N	301	429	86	HDPHOTO - LOSSLESS_WAVE	110	2.163610	-1.057827
160.jp2	24	RGB(A)	N	301	429	109	JP2 - LOSSLESS_WAVELET	203	1.347044	-0.170958
160.jls	24	RGB(A)	N	301	429	22	JLS - LOCO-1	22	1.449010	-2.040953
160.pcx	4	RGB(A)	N	301	429	1	PCX - RLE	1	0.475584	52.441601
160_deflate1.png	4	RGB(A)	N	301	429	1	PNG - DEFLATE_1	1	0.325230	67.476962
160.tga	8	RGB(A)	N	301	429	1	TGA - RLE	8	0.613783	4.827710
160_LZW.tiff	4	RGB(A)	N	301	429	1	TIFF - LZW	24	0.316842	2.846493
160_Packbits.tiff	4	RGB(A)	N	301	429	1	TIFF - PACKBITS	16	0.416343	3.647858
2007-08-18-TajemstvíKamenu.wdp	24	RGB(A)	N	397	496	148	HDPHOTO - LOSSLESS_WAVE	195	0.644483	0.182316
2007-08-18-TajemstvíKamenu.jp2	24	RGB(A)	N	397	496	328	JP2 - LOSSLESS_WAVELET	554	0.610918	0.070231
2007-08-18-TajemstvíKamenu.jls	24	RGB(A)	N	397	496	116	JLS - LOCO-1	92	0.564957	0.472873

Obr.45 Průběh provádění testů

Po provedení testů se zobrazí dialog s dotazem, zda se má uložit report do souboru formátu CSV (hodnoty oddělené středníkem). Pokud je zvolena možnost „Ano“, dojde k uložení výsledků testů. V průběhu testu se do složky pro dočasné soubory (Temp) ukládají průběžně získané informace do souboru formátu CSV. Toto řešení je využito kvůli případným pádům aplikace při testování - soubor lze kdykoliv ze složky Temp přesunout, a tak tedy nedojde ke ztrátě výsledků před havárií. Pojmenování tohoto souboru je ve formátu „GrafickyProhlizec\_<rok>-<měsíc>-<den>\_<hodina>-<minuta>-<sekunda>.csv“ (např. GrafickyProhlizec\_2012-3-23\_19-28-40).

Výsledný report, který je vidět na Obr.46 lze exportovat položkou „Export dat“. Rozdíl mezi tímto exportem a exportem po provedení testů je ten, že v tomto reportu může být uloženo i více reportů z předešlých testů, a nebo výsledky jednotlivých souborů, které se v kompletním testu neobjevily.

Prohlížení	Název	BH [bpp]	BP	OŠ	Šířka [pix]	Výška [pix]	DD [ms]	Typ K	DK [ms]	KP	ČEK [% / ms]
77	Fog_LZW.tif	32	FLOAT	A	1000	672	52	TIFF - LZW	107	0,001091	0,260711
80	Fog_Packbits.tiff	32	FLOAT	A	1000	672	8	TIFF - PACKBITS	71	1,009061	-0,012762
81	jilek_vytrvaly.wdp	24	RGB(A)	N	363	629	124	HDPHOTO - LOSSLESS_WAVE	172	0,258665	0,431008
82	jilek_vytrvaly.jp2	24	RGB(A)	N	363	629	172	JP2 - LOSSLESS_WAVELET	312	0,165929	0,267331
83	jilek_vytrvaly.js	24	RGB(A)	N	363	629	38	JLS - LOCO-I	38	0,208284	2,083463
84	jilek_vytrvaly.pcx	24	RGB(A)	N	363	629	32	PCX - RLE	31	0,327765	2,168500
85	jilek_vytrvaly_deflate1.png	24	RGB(A)	N	363	629	31	PNG - DEFLATE_1	47	0,210196	1,680435
86	jilek_vytrvaly.tga	24	RGB(A)	N	363	629	8	TGA - RLE	8	0,276158	9,048025
87	jilek_vytrvaly_LZW.tiff	24	RGB(A)	N	363	629	16	TIFF - LZW	47	0,261905	1,570416
88	jilek_vytrvaly_Packbits.tiff	24	RGB(A)	N	363	629	15	TIFF - PACKBITS	8	0,278011	9,024660
89	PIA12941.wdp	8	RGB(A)	A	1000	1000	296	HDPHOTO - LOSSLESS_WAVE	320	0,589563	0,128261
90	PIA12941.jp2	8	RGB(A)	A	1000	1000	577	JP2 - LOSSLESS_WAVELET	936	0,551463	0,047921
91	PIA12941.js	8	RGB(A)	A	1000	1000	132	JLS - LOCO-I	132	0,495987	0,381828
92	PIA12941.pcx	8	RGB(A)	A	1000	1000	39	PCX - RLE	16	0,842349	0,985319
93	PIA12941_deflate1.png	8	RGB(A)	A	1000	1000	47	PNG - DEFLATE_1	204	0,629309	0,181711
94	PIA12941.tga	8	RGB(A)	A	1000	1000	24	TGA - RLE	39	0,899046	0,258857
95	PIA12941_LZW.tiff	8	RGB(A)	A	1000	1000	31	TIFF - LZW	94	0,738517	0,278174
96	PIA12941_Packbits.tiff	8	RGB(A)	A	1000	1000	1	TIFF - PACKBITS	31	0,865999	0,432260
97	PIA_01.wdp	24	RGB(A)	N	150	729	78	HDPHOTO - LOSSLESS_WAVE	86	0,443719	0,646388
98	PIA_01.jp2	24	RGB(A)	N	150	729	94	JP2 - LOSSLESS_WAVELET	171	0,177378	0,481066
99	PIA_01.js	24	RGB(A)	N	150	729	22	JLS - LOCO-I	14	0,179208	5,862803
100	PIA_01.pcx	24	RGB(A)	N	150	729	1	PCX - RLE	8	0,079836	11,502047
101	PIA_01_deflate1.png	24	RGB(A)	N	150	729	8	PNG - DEFLATE_1	8	0,046838	11,914526
102	PIA_01.tga	24	RGB(A)	N	150	729	1	TGA - RLE	8	0,183489	10,206388
103	PIA_01_LZW.tiff	24	RGB(A)	N	150	729	16	TIFF - LZW	24	0,078984	3,837568
104	PIA_01_Packbits.tiff	24	RGB(A)	N	150	729	1	TIFF - PACKBITS	16	0,134554	5,409035
105	sandra.wdp	8	RGB(A)	A	150	179	15	HDPHOTO - LOSSLESS_WAVE	15	0,690695	2,062033
106	sandra.jp2	8	RGB(A)	A	150	179	15	JP2 - LOSSLESS_WAVELET	40	0,604928	0,987679

Obr.46 Report ve vytvořené aplikaci

Výsledný soubor lze následně spustit v tabulkovém editoru např. v programu Microsoft Excel nebo LibreOffice. Výsledek pak může vypadat jako na Obr.47.

1	Výsledky testů kompresních algoritmů	Grafický prohlížeč - Diplomová práce - Bc. Tomáš Vogetanz
2	Datum a čas:	27.3.2012 15:52
3		
4	Název	BH [bpp]
5	amonomarh(2)_RLE.bmp	8
6	amonomarh(2)_PIZ.exr	96
7	amonomarh(2)_ZIP.exr	96
8	amonomarh(2)_gif	8
9	amonomarh(2)_wdp	24
10	amonomarh(2)_jp2	24
11	amonomarh(2)_js	24
12	amonomarh(2)_pcx	24
13	amonomarh(2)_deflate1.png	24
14	amonomarh(2)_deflate1.png	24
15	amonomarh(2)_tga	24
16	amonomarh(2)_LZW.tiff	24
17	amonomarh(2)_Packbits.tiff	24
18	amonomarh(2)_CCITTFAX3.tiff	1
19	amonomarh(2)_CCITTFAX4.tiff	1
20	ccitt1_RLE.bmp	8
21	ccitt1_PIZ.exr	32
22	ccitt1_ZIP.exr	32
23	ccitt1_gif	1
24	ccitt1.wdp	1
25	ccitt1.jp2	8
26	ccitt1.js	8
27	ccitt1.pcx	1
28	ccitt1_deflate1.png	1
29	ccitt1_deflate1.png	1
30	ccitt1.tga	8
31	ccitt1_LZW.tiff	1
32	ccitt1_Packbits.tiff	1

Obr.47 Výsledky testů otevřené v aplikaci Microsoft Excel

## 5.1.2 Menu Obrázek

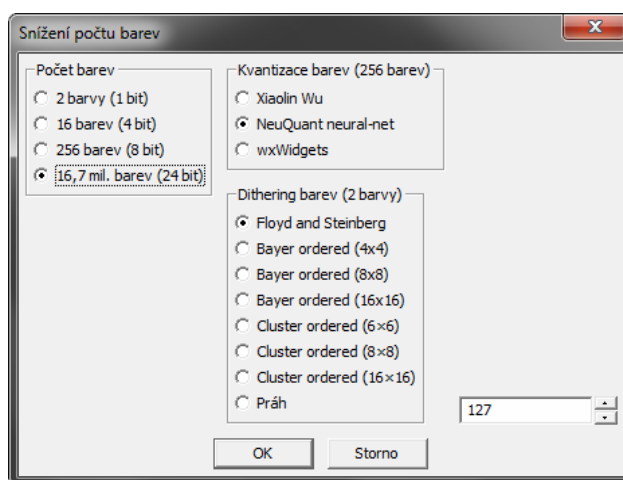
Práce s obrázky je převážně soustředěna na barevné prostory RGB, RGBA, Gray FLOAT, RGBF a RGBAF, ale je možné pracovat i s obrázky RGB16 a RGBA16.

V menu Obrázek se vyskytuje funkce horizontálního a vertikálního zrcadlení, inverze barev (nelze provést pro obrázky s barevným prostorem Gray FLOAT, RGBF a RGBAF), převod do odstínu šedi, snížení počtu barev obrázku a mapování barev (resp. tónování barev).

Výstupní bitová hloubka obrázku při převodu do odstínu šedi závisí na barevném prostoru vstupního obrázku - u barevného prostoru RGBF, RGBAF, RGB16 a RGBA16 dojde k převodu na barevný prostor Gray FLOAT (32 bitů), u barevných prostorů RGB a RGBA dochází k převodu na 8 bitový obrázek.

Po kliknutí na položku „Snížení počtu barev“ se objeví dialog zobrazený na *Obr.48*. Tento dialog umožňuje výběr počtu barev, na který se bude obrázek převádět a algoritmy, kterými se převod provede. Algoritmy se volí pro 2 a 256 barev, při převodu na 16 barev je vždy použita knihovna wxWidgets. Knihovna FreeImage sice také umožňuje kvantizaci na 16 barev, ale výsledná bitová hloubka obrázku je 8 bitů, a nikoliv 4 bity, jak by bylo vhodné - použitím knihovny wxWidgets byl tento problém vyřešen. Přebod na 16,7 mil. barev se provádí pro barevné prostory RGB16, RGBA16, Gray FLOAT, RGBF a RGBAF. Při snížení barev na 24 bitů je využito vybraného algoritmu v dialogu „EXR - mapování barev“.

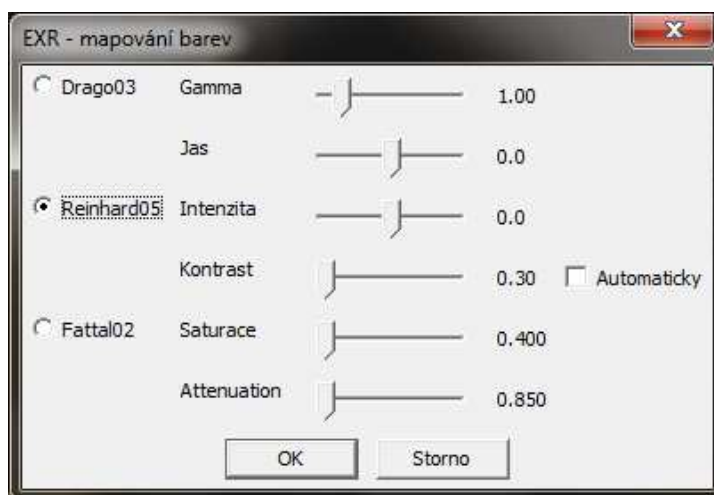
Převod se neprovede, pokud je zvolená bitová hloubka větší nebo rovna bitové hloubce vstupního obrázku, popř. je bitová hloubka rovna 1. Při provádění testů se u případných převodů barev (např. při ukládání do formátu GIF nebo CCITT FAX3) použijí aktuálně zvolené algoritmy v tomto dialogu.



*Obr.48 Dialog pro snížení počtu barev*

Položka „EXR - mapování barev“ slouží pro určení algoritmu, který provede převod barev z formátu obsahující vysoký dynamický rozsah barev (převážně určeno pro formát EXR, ale také pro TIFF a popř. i HDR a PFM). Mapování barev se využívá při zobrazování tohoto typu obrázku - na výchozí obrázek s vysokou dynamikou se aplikuje zvolená metoda a dojde k převodu na 24 bitovou hloubku, díky které lze obrázek vykreslit běžnými programovacími technikami.

Zobrazený dialog je vidět na *Obr.49*, ve kterém jsou vyplněny výchozí hodnoty, které se pro mapování barev používají. Každému typu algoritmu přísluší dva parametry, např. při volbě algoritmu Drago03 lze nastavit parametry Gamma a Jas, ostatní parametry v tu chvíli nemají na funkčnost vliv. Při mapování barev většinou dochází ke snížení počtu barev v obrázku.



*Obr.49 Dialog pro mapování barev formátu s vysokým dynamickým rozsahem*

### 5.1.3 Menu Nápověda

V menu „Nápověda“ je pouze položka s názvem „O aplikaci“. Tato položka obsahuje informace o podporovaných formátech, použitých knihovnách, konzolových aplikacích a také vysvětlivky ke zkratkám v hlavních sloupcích reportu. Význam těchto zkratek je také popsán v *Tab.34*.

Tab.34 Význam zkratek v reportu

Zkratka	Význam
Název	název souboru s obrázkem
BH [bpp]	bitová hloubka v bitech na pixel
BP	barevný prostor
OŠ	odstíny šedi (Ano/Ne)
Šířka [pix]	šířka obrázku v pixelech
Výška [pix]	výška obrázku v pixelech
DD [ms]	doba dekomprese v ms
Typ K	typ (výstupní) komprese
DK [ms]	doba komprese v ms
KP	kompresní poměr
ČEK [% / ms]	časová efektivita komprese (viz kapitola 5.1.1)

## 5.2 Popis zdrojových kódů aplikace

Projekt obsahuje následující soubory, které byly vytvořeny v aplikaci wxFormBuilder:

- `exr_mapovanibarev.fbp` - dialog pro položku „EXR - mapování barev“
- `gui.fbp` - grafické rozhraní hlavní aplikace
- `nastavenikompreseformatu.fbp` - dialog pro položku „Nastavení komprese formátů“
- `oaplikaci.fbp` - dialog pro položku „O aplikaci“
- `snizenipoctubarev.fbp` - dialog pro položku „Snížení počtu barev“
- `ukazatelstavu.fbp` - dialog s ukazatelem průběhu
- `vyberformatutestu.fbp` - dialog s výběrem formátů, které se budou testovat

Po vygenerování zdrojového kódu v jazyce C++ přes aplikaci wxFormBuilder vznikly stejnojmenné soubory s příponou `.h` (hlavička) a `.cpp` (zdrojový kód).

Dalšími soubory jsou `exr_mapovanibarev_main.h`, `exr_mapovanibarev_main.cpp`. Tyto dva zmíněné soubory implementují dialog pro položku „EXR - mapování barev“ a obslužné metody pro událost změny pozice posuvníků, ovlivňujících hodnoty parametrů algoritmů.

Konkrétní intervaly pro jednotlivé parametry jsou:

- Gamma -  $\langle 0,05 ; 8 \rangle$
- Jas -  $\langle -8 ; 8 \rangle$
- Intenzita -  $\langle -8 ; 8 \rangle$
- Kontrast -  $\langle 0,3 ; 1 \rangle$
- Saturace -  $\langle 0,4 ; 0,6 \rangle$
- Attenuation -  $\langle 0,8 ; 0,9 \rangle$

Soubor FreeImage.h umístěný ve složce h\_pomocne slouží jen jako pomocná knihovna, v aplikaci nemá žádné využití. Z této knihovny byly použity struktury, výčtové typy, informace o funkcích a některá makra pro implementaci třídy FreeImageRozhraniClass v souboru freeimagerozhrani.h.

Posledními čtyřmi soubory jsou main.h, main.cpp, obrazek.h a obrazek.cpp. Soubory main obsahují implementaci hlavního grafického rozhraní a jsou jádrem celé aplikace. V souborech obrazek je vytvořena třída Obrazek, která umožňuje načtení a uložení souborů formátu BMP (1, 4, 8, 24 bitů) a PCX (1, 4, 8, 24 bitů - u 24 bitů pouze načtení).

### 5.2.1 Třída Obrazek

Diagram této třídy a její asociace zobrazuje Příloha P III. Třída Obrazek je implementována v souborech obrazek.h a obrazek.cpp. Mimo ní jsou zde vytvořeny struktury hlaviček formátu BMP a PCX a výčtový typ pro určení formátu, který byl načten.

Třída obsahuje atributy s informacemi o formátu obrázku, bitové hloubce, šířce, výšce apod. a samozřejmě také samotná obrazová data. Obrazová data jsou uložena ve formátu BMP. Když se tedy načte formát PCX, tak jsou poté data dekodována do formátu BMP.

Metoda pro načtení (NacistDataZeSouboru) a dekodování (Dekodovat) obrázku jsou ve třídě odděleny kvůli eliminaci vlivu času načítání souboru z pevného disku při měření doby dekomprese. Kvůli stejnému důvodu je také odděleno kódování (Kodovat) a uložení souboru (UlozitDataDoSouboru). Kvůli většímu komfortu zde je však také implementována metoda, která provede obě operace najednou - tedy načtení i dekodování (NacistADekodovatZeSouboru) resp. kódování i uložení (KodovatAUlozitDoSouboru).

Dekódování je provedeno způsobem, který umožňuje použít stejný algoritmus pro obrázky s jednou bitovou rovinou a bitovou hloubkou 1, 4, 8, a také se třemi bitovými rovinami a bitovou hloubkou 8 (24 bitů) a navíc není nutné provést po dekodování vertikální zrcadlení nebo převod na barevný prostor BGR. Toto je možné díky následujícímu výpočtu indexu pole, na který se má aktuálně dekodovaná hodnota vložit:

$$I = (V - R - 1) \cdot (BnR - DBR) \cdot BR + UBR \cdot BR + BR - 1 - AR + PDB \cdot (V - R - 1)$$

I - index, na který se hodnota uloží (prvním indexem je 0)

V - výška obrázku v pixelech (vypočítaná z hlavičky PCX)

R - aktuálně zpracovávaný řádek obrázku

BnR - počet Byte na řádek (z hlavičky PCX)

DBR - doplnění Bytu na řádek pro PCX

BR - počet bitových rovin (z hlavičky PCX)

UBR - počet již zpracovaných Bytů z aktuální dekodované roviny

AR - aktuálně dekodovaná rovina (1. rovina je označena číslem 0)

PDB - počet doplňujících Byte pro BMP (aby byla velikost řádku dělitelná čtyřmi)

Proměnná DBR se určí následujícím vzorcem:

$$DBR = BnR - \frac{S \cdot BH}{8}$$

BnR - počet Byte na řádek (z hlavičky PCX)

S - šířka obrázku v pixelech

BH - bitová hloubka obrázku (počet bitů na pixel z hlavičky PCX)

Proměnnou PDB lze vypočítat takto:

$$PDB = \text{ceil}\left(\frac{S \cdot BH}{8 \cdot 4}\right) \cdot 4 - \frac{S \cdot BH}{8}$$

S - šířka obrázku v pixelech

BH - bitová hloubka obrázku (počet bitů na pixel pro BMP)

ceil - funkce pro zaokrouhlení desetinného čísla vždy na vyšší hodnotu

Kódování bylo naprogramováno jen pro bitovou hloubku 1, 4 a 8 bitů. PCX s 8 bity na pixel a 3 bitovými rovinami se v aplikaci ukládá pomocí knihovny wxWidgets.

### 5.2.2 Třída FreeImageRozhraniClass

Diagram této třídy a její asociace zobrazuje Příloha P IV. Třída FreeImageRozhraniClass je implementována v souboru freeimagerozhrani.h. Jedná se o rozhraní mezi knihovnou FreeImage a aplikací. Mimo ní jsou zde ze souboru FreeImage.h převedeny struktury, výčtové typy, makra a definovány datové typy ukazatelů na konkrétní typ funkce pro načtení těchto funkcí z dynamické knihovny FreeImage.dll.

V této třídě patří mezi hlavní atributy ukazatel na aktuálně načtený obrázek, jeho formát, handle DLL knihovny a informace o typu kvantizačního, mapovacího a dithering algoritmu, které jsou využívány metodou pro snížení počtu barev v obrázku.

Zjednodušený příklad načtení dynamické knihovny a funkce v této knihovně je ukázán v následujícím zdrojovém kódu:

```
1 #include <stdio.h>
2 #include <windows.h>
3 //definování datového typu - ukazatel na funkci Load
4 typedef FIBITMAP* (__stdcall * FREEIMAGE_LOAD)(FREE_IMAGE_FORMAT
  fif, const char *filename, int flags);
5 //ukazatel na funkci Load
6 FREEIMAGE_LOAD FreeImage_Load;
7 //handle na dll knihovnu
8 HINSTANCE m_hDll;
9 //pokud se zdaří načtení dll knihovny
10 if((m_hDll = LoadLibrary(_("FreeImage.dll"))) != NULL)
11 {
12     //dojde k pokusu o načtení funkce s daným názvem (přesný název
  funkce byl zjištěn otevřením souboru FreeImage.dll v hex-editoru)
13     FreeImage_Load = (FREEIMAGE_LOAD)GetProcAddress(m_hDll,
  "_FreeImage_Load@12");
14     //pokud se funkci nepodařilo načíst vypíše se chyba
15     if(FreeImage_Load == NULL)
16     {
17         printf("chyba pri nacistani funkce FreeImage_Load\n");
18     }
}
```

Třída `FreeImageRozhraniClass` obsahuje metody pro práci s obrázky, které využívají načtených funkcí z dynamické knihovny. K nejdůležitějším patří metody načtení (`NacistObrazek`) a uložení (`UlozitObrazek`) obrázku, konverze barevného prostoru (`PrevodBarevnehoProstoru`), dále zjištění bitové hloubky, šířky, výšky, barevného prostoru, převod na odstíny šedé, snížení počtu barev, zrcadlení, inverze barev a nakonec i převod bitmapy využívané knihovnou `FreeImage` (struktura `FIBITMAP`) na objekt třídy `wxImage` (`FIBITMAP_na_WxImage`). Poslední metoda je naprogramována kvůli zobrazení obrázku v prohlížečím okně aplikace pomocí knihovny `wxWidgets`.

Metodu pro převod barevného prostoru (resp. převod bitové hloubky) je dobré použít před uložením samotného souboru. Dochází zde totiž ke snížení bitové hloubky v závislosti na zvoleném formátu, který má být pro uložení použit. Pokud se tato metoda před uložením neprovede, může dojít k vytvoření souboru s nulovou délkou - jinými slovy uložení souboru se nezdaří.

### 5.2.3 Třída `MainFrame`

Diagram této třídy a její asociace zobrazuje Příloha P V. Ve třídě `MainFrame` (soubor `main.h` a `main.cpp`) je implementováno základní grafické rozhraní a obslužné metody událostí. Základem jsou tedy obslužné metody pro kliknutí na položky v menu - např. `OnSouborOtevritClick`, `OnSouborUlozitJakoClick`, `OnSouborNastaveniKompreseClick`, `OnSouborProvestTestyClick` apod. Dále je zde také obslužná metoda pro vykreslování obrázků do okna s možností skrolování (`OnScrolledWindowProhlizeniPaint`). Aby se obrázek zbytečně nevykresloval celý, byly naprogramovány metody pro výpočet oblasti, kterou je nutné vykreslit (`VypocetPocatkuAKonceBitmapy`, `VypocetPosunuPozice`). Před vykreslením pak dojde k "vystřížení" této vypočtené oblasti a následně se zobrazuje jen tato část bitmapy - tím se sníží zátěž procesoru při skrolování u obrázků s vysokým rozlišením.

Při načítání obrázků je nejdříve využita knihovna `FreeImage`. Pokud se knihovně nepovede načíst vstupní soubor, dojde k pokusu o načtení souboru `PCX` (pomocí objektu třídy `Obrazek`). Pokud i tento pokus selže testuje se přípona souboru, a pokud je totožná s příponami pro formát `HD Photo`, `JBIG` nebo `JPEG-LS`, tak se vytvoří proces a spustí se příslušná konzolová aplikace (provede se metoda `ProcesSpustit`). U uložení se jedná o něco podobného, jen je zde specifikován výstupní formát, takže je vždy možné určit, která

komponenta se o uložení postará. Po uložení obrázku je určen kompresní poměr a časová efektivita komprese.

Pro spuštění konzolových aplikací se vytvoří synchronní proces a je zjištěno jeho PID. Synchronní znamená, že aplikace čeká na dokončení tohoto procesu. Poté se z výchozí zprávy aplikace zjistí informace o době komprese, popř. dekomprese. Soubory potřebné pro převod formátů pomocí této aplikace jsou ukládány do složky dočasných souborů (Temp). Ve chvíli, kdy jsou soubory nepotřebné dojde k jejich smazání. Názvy těchto souborů se vytvářejí z aktuálního data a času.

Pro výpočet kompresního poměru se používá metoda `ZjistitKompresniPomer` a pro výpočet časové efektivity metoda `ZjistitCasovouEfektivitu`. Pro výpočet kompresního poměru se stanoví velikost uloženého souboru a dále se podle výšky šířky a bitové hloubky vypočte velikost, kterou by měl soubor BMP v nekomprimované podobě - počítá se zde i s hlavičkou souboru a informační hlavičkou, zarovnáním na řádky dělitelné čtyřmi, a případnou paletou barev. Vzorec pro výpočet velikosti souboru BMP vypadá takto:

$$VBMP = \text{ceil}\left(\frac{S \cdot V \cdot BH}{8}\right) + PDB \cdot V + VH + VP$$

VBMP - velikost souboru BMP

S - šířka obrázku v pixelech

V - výška obrázku v pixelech

BH - bitová hloubka obrázku

PDB - počet doplňujících Byte pro BMP (aby byla velikost řádku dělitelná čtyřmi)

VH - velikost hlavičky a informační hlavičky BMP V3 (tzn. 54 Bytů)

VP - velikost palety (u obrázků s bitovou hloubkou větší než 8 je velikost palety 0)

ceil - funkce pro zaokrouhlení desetinného čísla vždy na vyšší hodnotu

Pro uložení exportovaných údajů se využívá formátu CSV s oddělením hodnot pomocí středníku (tzn. „;“) a oddělením řádků pomocí znaků s hodnotou v ASCII tabulce 0Dh a 0Ah (Aplikace Microsoft Excel načte i soubor bez znaku 0Dh). V programovém kódu se tyto znaky do textu zapíší jako „\r\n“.

Formát CSV byl využit díky jednoduchosti, dostatečným možnostem pro zvolený účel (včetně možnosti jednoduchého načtení v tabulkové aplikaci) a jen malým omezením - v textu nelze použít znaky „;“ „\r“ „\n“.

## 6 TESTOVÁNÍ KOMPRESNÍCH ALGORITMŮ

Testování bylo prováděno na zařízení s následujícími parametry:

- Intel® Core™ i5-430M 2,26 GHz (Turbo 2,53 GHz); 2 jádra; 4 vlákna; 3 MB Cache; 1066 MHz; 32 nm
- 4 GB DDR3 SDRAM; 1066 MHz
- HDD SATA 5400 ot/min; vyrovnávací paměť 8192 kB; doba přístupu 18,4 ms; průměrná přenosová rychlost 65 MB/s
- Windows® 7 Professional 64bit

Testované soubory dat byly rozděleny na 12 kategorií, které jsou hodnoceny odděleně. Výsledky testů jsou uvedeny v tabulce a grafu pro každou kategorii a jsou seřazeny vzestupně podle průměru hodnot kompresního poměru. Vysvětlivky pro zkratky v tabulkách jsou v *Tab.35*.

*Tab.35 Význam zkratek v tabulkách s výsledky*

Zkratka	Význam
Typ K	typ komprese
DD [ms]	doba dekomprese v ms
DK [ms]	doba komprese v ms
$KP_{\phi}$	průměr kompresních poměrů
$KP_{med}$	medián kompresních poměrů
$KP_{min}$	minimální kompresní poměr
$KP_{max}$	maximální kompresní poměr
$\sigma_{KP}$	směrodatná odchylka kompresních poměrů
ČEK [%/ms]	časová efektivita komprese

U každé kategorie je vybrán průměrný testovaný obrázek, a to z hlediska průměru kompresních poměrů a průměru počtu pixelů testovaných obrázků. Tyto obrázky byly vybrány převážně podle kompresního poměru nejlépe hodnoceného kompresního algoritmu, ale sekundárním požadavkem byly kompresní poměry ostatních typů komprese.

Obrázky využití v testech byly vyhledány na webech: [62] [63] [64] [65] [66] [67] [68] [69]. Celkově bylo pro testování využito 10 842 obrázků. Příloha P XVIII zobrazuje přehled nejlepších výsledků v testech.

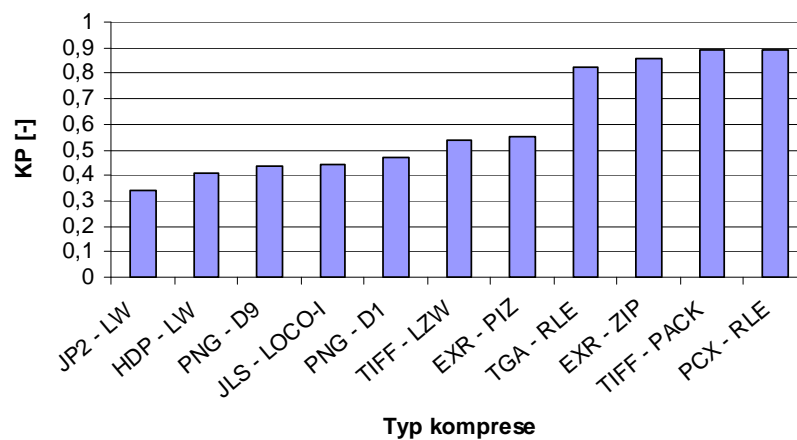
## 6.1 Fotografie

### 6.1.1 Fotografie formátu JPEG

V této kategorii jsou testovány fotografie, které byly původně uloženy ve formátu JPEG - před testem na ně tedy byla aplikována ztrátová komprese. Celkem bylo testováno 2839 fotografií a průměrný počet pixelů jedné fotografie se pohybuje okolo 1 893 965 pixelů. Výsledky testů jsou uvedeny v *Tab.36* a grafické zobrazení průměru kompresních poměrů je na *Obr.50*.

*Tab.36 Výsledky testů pro fotografie formátu JPEG*

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
JP2 - LW	1423	2009	0,342	0,354	0,037	0,826	0,139	0,033
HDP - LW	717	817	0,407	0,414	0,073	0,850	0,127	0,073
PNG - D9	131	3464	0,438	0,451	0,043	0,919	0,165	0,016
JLS - LOCO-I	453	383	0,444	0,458	0,030	0,927	0,159	0,145
PNG - D1	142	402	0,469	0,499	0,060	0,882	0,161	0,132
TIFF - LZW	85	147	0,534	0,549	0,078	1,213	0,194	0,317
EXR - PIZ	158	336	0,550	0,572	0,067	1,019	0,167	0,134
TGA - RLE	20	54	0,822	0,941	0,080	1,270	0,238	0,329
EXR - ZIP	249	2101	0,857	0,927	0,081	1,566	0,315	0,007
TIFF - PACK	21	58	0,888	1,006	0,082	1,329	0,252	0,194
PCX - RLE	125	228	0,889	0,990	0,064	1,640	0,274	0,049



*Obr.50 Graf výsledku testů pro fotografie formátu JPEG*

Na fotografie původně uložené ve formátu JPEG se nejlépe hodí použít formát JPEG 2000 s vlnkovou kompresí. Následující vlnková komprese formátu HD Photo dosahovala o asi 6 procent horší výsledky. JPEG 2000 exceluje nejenom z hlediska průměru kompresních poměrů, ale i mediánu, maximálního a minimálního kompresního poměru - u mediánu a maxima dosáhl nejlepších hodnot, u minima byl tento formát horší jen o 0,7 % než formát JPEG-LS. Jediným negativem JPEG 2000 je dlouhá doba komprese. Pokud je kladen důraz spíše na tento parametr a následně pak na kompresní poměr, tak je vhodnější použít HD Photo, popř. i JPEG-LS nebo PNG-DEFLATE\_1.

Pouze prvních 5 kompresí uvedených v *Tab.36* nedosáhly záporné komprese - konkrétně JPEG 2000, HD Photo, PNG-DEFLATE\_9, JPEG-LS a PNG-DEFLATE\_1. V porovnání s těmito formáty nejsou ostatní formáty vhodné pro ukládání fotografií původně uložených ve formátu JPEG.

Příloha P VI zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 1600×1200 (1 920 000 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.37*.

*Tab.37 Výsledky testů pro vybranou fotografii formátu JPEG*

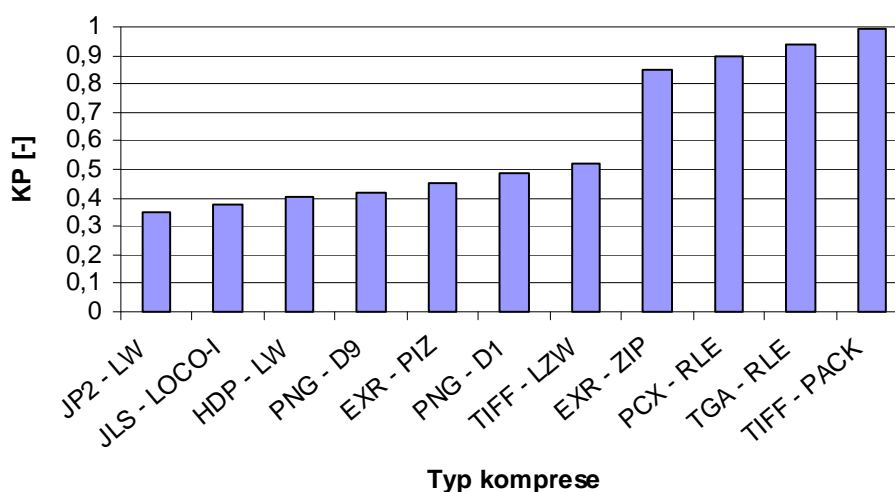
Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
JP2 - LW	1178	1935	0,318	0,035
HDP - LW	710	804	0,400	0,075
PNG - D9	125	3955	0,425	0,015
JLS - LOCO-I	435	382	0,415	0,153
PNG - D1	140	375	0,480	0,139
TIFF - LZW	93	157	0,555	0,284
EXR - PIZ	164	351	0,546	0,129
TGA - RLE	31	47	0,831	0,360
EXR - ZIP	242	2052	0,935	0,003
TIFF - PACK	24	70	0,997	0,005
PCX - RLE	125	218	0,897	0,047

### 6.1.2 Fotografie formátu RAW

Zdrojem testovaných subjektů jsou fotografie pořízené v nezkomprimovaném formátu příslušným typem fotoaparátu - konkrétně formáty CR2, NEF, ORF, RAF. Celkem bylo testováno 90 fotografií s průměrným počtem pixelů 9 083 009. Výsledky testů jsou uvedeny v *Tab.38* a grafické zobrazení průměru kompresních poměrů je na *Obr.51*.

*Tab.38* Výsledky testů pro fotografie formátů RAW

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
JP2 - LW	5383	8492	0,347	0,317	0,207	0,598	0,095	0,008
JLS - LOCO-I	2067	1745	0,375	0,358	0,190	0,736	0,118	0,036
HDP - LW	3437	3881	0,404	0,380	0,303	0,634	0,081	0,015
PNG - D9	678	34073	0,419	0,395	0,243	0,773	0,111	0,002
EXR - PIZ	743	1670	0,455	0,427	0,257	0,812	0,120	0,033
PNG - D1	723	1933	0,484	0,461	0,314	0,784	0,105	0,027
TIFF - LZW	362	661	0,523	0,491	0,267	0,967	0,158	0,072
EXR - ZIP	1318	13775	0,847	0,814	0,542	1,300	0,188	0,001
PCX - RLE	619	1017	0,894	0,909	0,589	1,137	0,107	0,010
TGA - RLE	77	278	0,942	0,957	0,746	1,002	0,060	0,021
TIFF - PACK	96	286	0,994	1,005	0,826	1,009	0,030	0,002



*Obr.51* Graf výsledku testů pro fotografie formátu RAW

Výsledky těchto testů se převážně u nejlepších typů kompresí velmi podobají výsledkům testů fotografií JPEG. A to jak v pořadí, tak v hodnotách kompresních poměrů. Výrazné zlepšení kompresních poměrů lze pozorovat u formátů JPEG-LS (asi 7 %) a EXR-PIZ (asi 10 %) a výrazné zhoršení u formátů TGA (asi 12 %) a TIFF-PACKBITS (asi 11 %).

Formáty, u kterých nedošlo k záporné kompresi jsou JPEG 2000, JPEG-LS, HD Photo, PNG-DEFLATE\_9, EXR-PIZ, PNG-DEFLATE\_1 a TIFF-LZW. Záporná komprese se u tohoto typu fotografií již neobjevuje v takové míře jako u fotografií JPEG.

Jako nejlepší se jeví použití formátu JPEG 2000 s vlnkovou kompresí, který dosáhl nejlepšího průměru kompresních poměrů, mediánu a maximálního kompresního poměru. U minimálního kompresního poměru byl lepší o 1 % formát JPEG-LS. Pokud by byla na prvním místě potřeba využití rychlé komprese, je vhodnější použít formát JPEG-LS, který komprimuje obrázky asi 5-krát rychleji.

Příloha P VII zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 2760×3680 (10 156 800 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.39*.

*Tab.39 Výsledky testů pro vybranou fotografii formátu RAW*

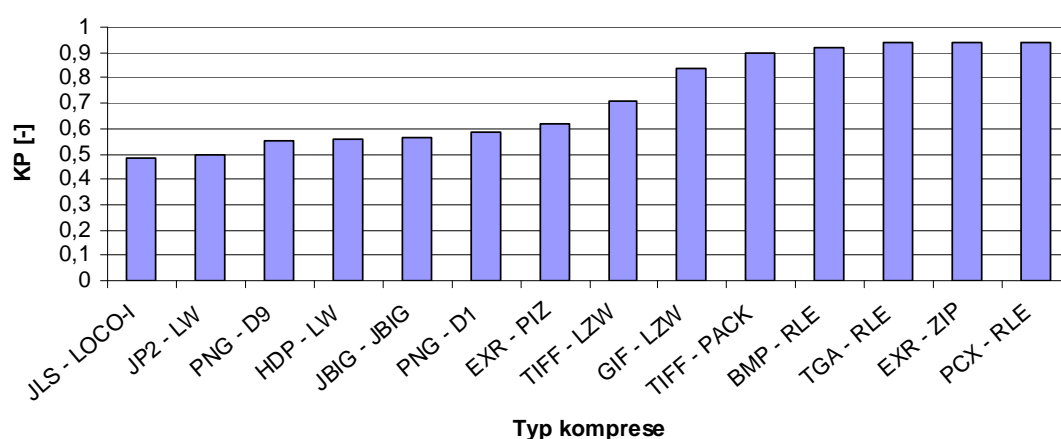
Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
JP2 - LW	6498	10554	0,365	0,006
JLS - LOCO-I	2463	2074	0,383	0,030
HDP - LW	3986	4454	0,419	0,013
PNG - D9	804	22012	0,436	0,003
EXR - PIZ	917	1906	0,481	0,027
PNG - D1	858	2254	0,508	0,022
TIFF - LZW	437	796	0,575	0,053
EXR - ZIP	1512	15778	0,915	0,001
PCX - RLE	695	1186	0,929	0,006
TGA - RLE	78	296	0,983	0,006
TIFF - PACK	125	312	1,002	-0,001

### 6.1.3 Fotografie v odstínech šedi

Tato sekce se zabývá fotografiemi v odstínech šedi. Celkem bylo testováno 665 fotografií s průměrným počtem pixelů 1 855 096. Výsledky testů jsou uvedeny v *Tab.40* a grafické zobrazení průměru kompresních poměrů je na *Obr.52*.

*Tab.40 Výsledky testů pro fotografie v odstínech šedi*

Typ K	DD [ms]	DK [ms]	KP <sub>0</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
JLS - LOCO-I	125	110	0,483	0,485	0,007	0,888	0,139	0,469
JP2 - LW	426	683	0,496	0,493	0,008	0,898	0,139	0,074
PNG - D9	38	1170	0,552	0,562	0,004	0,906	0,139	0,038
HDP - LW	257	283	0,558	0,552	0,134	0,920	0,125	0,156
JBIG - JBIG	620	599	0,566	0,581	0,006	0,985	0,159	0,072
PNG - D1	41	152	0,586	0,597	0,011	0,913	0,131	0,272
EXR - PIZ	75	140	0,621	0,630	0,066	0,985	0,151	0,271
TIFF - LZW	29	69	0,711	0,714	0,029	1,267	0,196	0,420
GIF - LZW	158	218	0,837	0,854	0,026	1,306	0,195	0,075
TIFF - PACK	10	33	0,898	0,960	0,025	1,008	0,141	0,310
BMP - RLE	23	18	0,919	0,988	0,020	1,023	0,137	0,450
TGA - RLE	17	50	0,937	1,004	0,026	1,077	0,150	0,127
EXR - ZIP	78	532	0,938	0,959	0,023	1,539	0,252	0,012
PCX - RLE	42	16	0,938	0,972	0,038	1,750	0,163	0,377



*Obr.52 Graf výsledku testů pro fotografie v odstínech šedi*

Nejlepší kompresí pro fotografie v odstínech šedi je komprese LOCO-I formátu JPEG-LS. Nejlepších výsledků dosáhl ve všech parametrech kromě minimálního kompresního poměru, kde však dosáhl nižší hodnoty jen o 0,3 % než komprese PNG-DEFLATE\_9. Nejlepší volba z hlediska rychlosti komprese je rovněž formát JPEG-LS.

Formáty, u kterých nedošlo k záporné kompresi jsou JPEG-LS, JPEG 2000, PNG-DEFLATE\_9, HD Photo, JBIG, PNG-DEFLATE\_1 a EXR-PIZ.

Příloha P VIII zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 1400×1400 (1 960 000 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.41*.

*Tab.41 Výsledky testů pro vybranou fotografii v odstínech šedi*

Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
JLS - LOCO-I	140	116	0,545	0,392
JP2 - LW	593	959	0,563	0,046
PNG - D9	31	289	0,603	0,137
HDP - LW	280	304	0,639	0,119
JBIG - JBIG	639	640	0,620	0,059
PNG - D1	31	180	0,625	0,209
EXR - PIZ	93	164	0,655	0,210
TIFF - LZW	48	70	0,824	0,251
GIF - LZW	195	258	0,846	0,060
TIFF - PACK	1	78	0,807	0,247
BMP - RLE	8	16	0,808	1,201
TGA - RLE	16	39	0,832	0,430
EXR - ZIP	86	429	1,040	-0,009
PCX - RLE	39	16	0,808	1,198

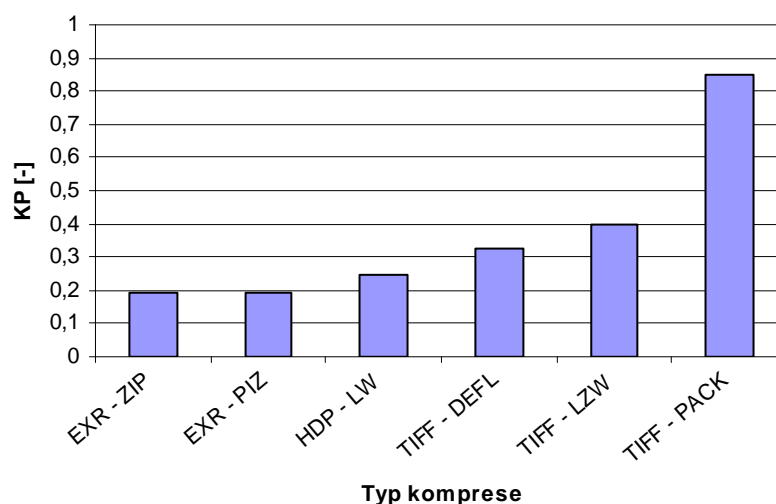
#### 6.1.4 Fotografie s vysokou dynamikou barev

V této kategorii byly testovány fotografie a formáty podporující barevný prostor RGBF, resp. RGBAF (u formátu HD Photo jediná možná varianta) - tedy 96 nebo 128 bitovou hloubku. Celkem bylo testováno 66 fotografií s průměrným počtem pixelů 815 251.

Výsledky testů jsou uvedeny v *Tab.42* a grafické zobrazení průměru kompresních poměrů je na *Obr.53*.

*Tab.42 Výsledky testů pro fotografie s vysokou dynamikou barev*

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
EXR - ZIP	105	566	0,191	0,186	0,003	0,398	0,109	0,143
EXR - PIZ	94	177	0,194	0,174	0,011	0,398	0,096	0,457
HDP - LW	371	397	0,250	0,252	0,071	0,413	0,082	0,189
TIFF - DEFL	100	1134	0,328	0,319	0,008	0,588	0,172	0,059
TIFF - LZW	104	224	0,400	0,394	0,070	0,675	0,177	0,268
TIFF - PACK	24	99	0,849	0,973	0,317	1,020	0,199	0,152



*Obr.53 Graf výsledku testů pro fotografie s vysokou dynamikou barev*

Z výsledků není možné určit jednoznačného vítěze. Kompresní algoritmy ZIP a PIZ formátu EXR si jsou kompresními poměry velmi podobné. S přihlédnutím k rychlosti komprese je lepší použít formát EXR s kompresí PIZ.

Vzhledem k možnosti dosažení záporné komprese kompresního algoritmu PACKBITS formátu TIFF a také vzhledem ke špatnému kompresnímu poměru není vhodné tuto

kompresi používat - všechny ostatní kompresní algoritmy jsou v tomto ohledu minimálně dvojnásobně lepší.

Příloha P IX zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 1214×732 (888 648 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.43*.

*Tab.43 Výsledky testů pro vybranou fotografii s vysokou dynamikou barev*

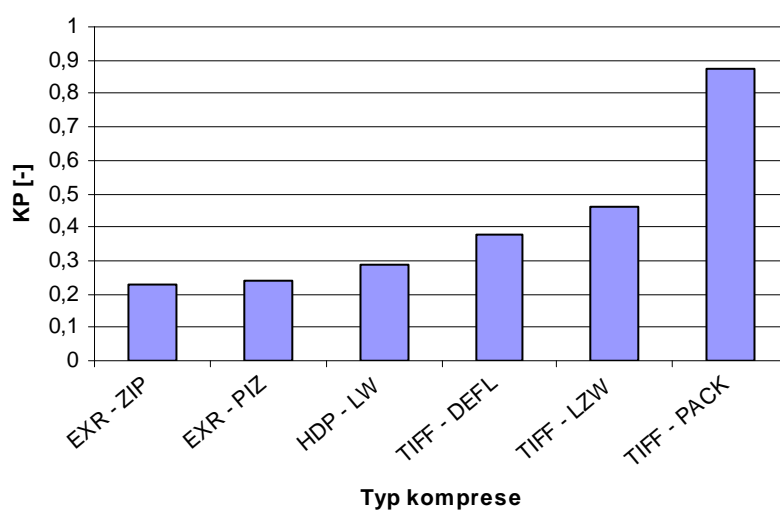
Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
EXR - ZIP	145	834	0,253	0,090
EXR - PIZ	107	276	0,234	0,278
HDP - LW	449	465	0,240	0,163
TIFF - DEFL	142	1900	0,429	0,030
TIFF - LZW	161	327	0,484	0,158
TIFF - PACK	23	157	1,010	-0,006

### 6.1.5 Fotografie s vysokou dynamikou barev v odstínech šedi

V této kategorii byly testovány fotografie a formáty podporující barevný prostor FLOAT (někdy též nazýván GRAY FLOAT) - tedy 32 bitovou hloubku. Celkem bylo testováno 72 fotografií s průměrným počtem pixelů 787 691. Výsledky testů jsou uvedeny v *Tab.44* a grafické zobrazení průměru kompresních poměrů je na *Obr.54*.

*Tab.44 Výsledky testů pro fotografie s vysokou dynamikou barev v odstínech šedi*

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
EXR - ZIP	32	153	0,227	0,251	0,007	0,429	0,110	0,505
EXR - PIZ	53	91	0,237	0,250	0,033	0,448	0,101	0,838
HDP - LW	129	139	0,288	0,298	0,101	0,401	0,075	0,512
TIFF - DEFL	37	298	0,379	0,425	0,014	0,589	0,165	0,208
TIFF - LZW	37	101	0,459	0,510	0,052	0,672	0,181	0,535
TIFF - PACK	9	61	0,874	1,008	0,337	1,023	0,191	0,206



Obr.54 Graf výsledku testů pro fotografie s vysokou dynamikou barev v odstínech šedi

Nejlepší kompresí pro tuto kategorii je komprese ZIP formátu EXR, kterou ale následuje komprese PIZ stejného formátu - rozdíl je u kompresního poměru pouze 1 %. Při nutnosti využití rychlé komprese je lepší použít kompresi PIZ.

Ze stejného důvodu, který byl zmíněn v kapitole 6.1.4 není vhodné využívat formátu TIFF s kompresí PACKBITS.

Příloha P X zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 900×900 (810 000 pixelů). Výsledky testu tohoto obrázku ukazuje Tab.45.

Tab.45 Výsledky testů pro vybranou fotografii s vysokou dynamikou barev v odstínech šedi

Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
EXR - ZIP	34	197	0,269	0,371
EXR - PIZ	57	94	0,267	0,780
HDP - LW	134	147	0,333	0,454
TIFF - DEFL	39	317	0,446	0,175
TIFF - LZW	36	161	0,513	0,302
TIFF - PACK	7	45	0,802	0,441

## 6.2 Obrázky

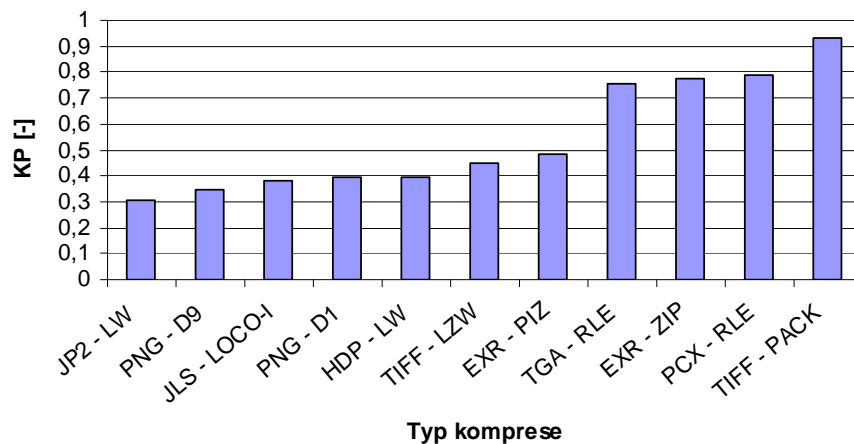
V této sekci byly testovány obrázky vytvořené pomocí PC (např. grafickým editorem).

### 6.2.1 Obrázky ve 24 bitové hloubce

Celkem bylo testováno 1 936 obrázků s průměrným počtem pixelů 1 336 500. Výsledky testů jsou uvedeny v *Tab.46* a grafické zobrazení průměru kompresních poměrů je na *Obr.55*.

*Tab.46 Výsledky testů pro obrázky ve 24 bitové hloubce*

Typ K	DD [ms]	DK [ms]	KP <sub>0</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
JP2 - LW	769	1257	0,308	0,284	0,001	0,898	0,122	0,055
PNG - D9	84	3649	0,350	0,333	0,002	0,921	0,162	0,018
JLS - LOCO-I	304	256	0,378	0,361	0,002	0,932	0,145	0,243
PNG - D1	91	251	0,395	0,392	0,006	0,885	0,155	0,241
HDP - LW	494	568	0,397	0,363	0,038	0,987	0,125	0,106
TIFF - LZW	59	108	0,451	0,433	0,022	1,515	0,188	0,509
EXR - PIZ	116	242	0,485	0,477	0,016	1,004	0,155	0,213
TGA - RLE	15	43	0,758	0,814	0,010	1,388	0,224	0,564
EXR - ZIP	174	1467	0,776	0,799	0,011	1,601	0,283	0,015
PCX - RLE	91	151	0,792	0,847	0,013	3,858	0,281	0,137
TIFF - PACK	16	52	0,931	1,003	0,021	1,469	0,182	0,133



*Obr.55 Graf výsledku testů pro obrázky ve 24 bitové hloubce*

Nejlepším formátem je pro tuto kategorii JPEG 2000, který z hlediska kompresního poměru porazil všechny formáty. Jen u maximálního kompresního poměru byla zaznamenána asi o 1 % lepší hodnota u komprese PNG-DEFLATE\_1. Z pohledu rychlosti komprese je možné využít formát JPEG-LS nebo PNG-DEFLATE\_1.

V *Tab.46* a *Tab.36* se objevuje jedna zajímavá zvláštnost, a to že komprese DEFLATE s úrovní 1 může u některých souborů dosáhnout lepšího kompresního poměru než komprese DEFLATE s úrovní 9 (viz hodnoty maximálních kompresních poměrů těchto dvou kompresí). Tam, kde komprese DEFLATE\_9 dosáhla výsledku 0,921 (tedy svého nejhoršího kompresního poměru v *Tab.46*), tam dosáhla komprese DEFLATE\_1 hodnoty 0,847. Zde tedy neplatí vždy pravidlo, čím větší zvolená úroveň tím lepší komprese.

Formáty, u kterých nedošlo k záporným kompresím byly JPEG 2000, PNG-DEFLATE\_9, JPEG-LS, PNG-DEFLATE\_1 a HD Photo. Přestože se komprese TIFF-PACKBITS svým průměrem kompresních poměrů nejvíce blíží trendu záporné komprese, nejhorší maximální kompresní poměr dosáhl formát PCX s kompresí RLE (3,858).

Příloha P XI zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 1600×900 (1 440 000 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.47*.

*Tab.47 Výsledky testů pro vybraný obrázek ve 24 bitové hloubce*

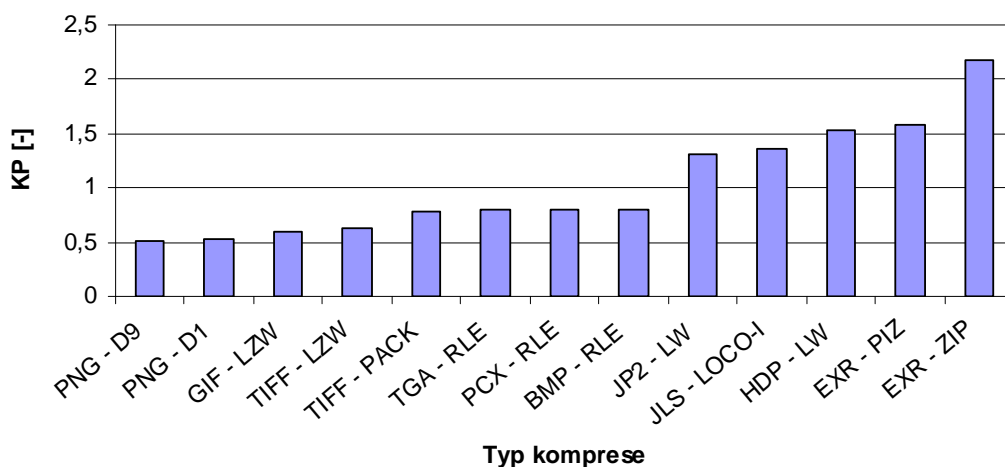
Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
JP2 - LW	864	1450	0,301	0,048
PNG - D9	87	3434	0,353	0,019
JLS - LOCO-I	302	256	0,342	0,257
PNG - D1	93	255	0,393	0,238
HDP - LW	513	584	0,356	0,110
TIFF - LZW	58	115	0,412	0,511
EXR - PIZ	117	241	0,435	0,235
TGA - RLE	16	42	0,740	0,620
EXR - ZIP	175	1486	0,709	0,020
PCX - RLE	92	133	0,715	0,214
TIFF - PACK	15	44	0,827	0,393

### 6.2.2 Obrázky v 8 bitové hloubce

Celkem bylo testováno 1 512 obrázků s průměrným počtem pixelů 1 215 197. Výsledky testů jsou uvedeny v *Tab.48* a grafické zobrazení průměru kompresních poměrů je na *Obr.56*.

*Tab.48 Výsledky testů pro obrázky v 8 bitové hloubce*

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
PNG - D9	14	318	0,503	0,531	0,007	0,946	0,212	0,156
PNG - D1	16	54	0,526	0,559	0,015	0,947	0,199	0,879
GIF - LZW	89	123	0,593	0,625	0,008	1,298	0,236	0,331
TIFF - LZW	17	44	0,624	0,646	0,028	1,606	0,236	0,860
TIFF - PACK	8	25	0,782	0,845	0,028	2,017	0,252	0,883
TGA - RLE	13	33	0,792	0,868	0,025	1,075	0,253	0,632
PCX - RLE	29	11	0,794	0,841	0,040	1,299	0,269	1,806
BMP - RLE	23	14	0,797	0,886	0,025	1,044	0,230	1,477
JP2 - LW	834	1370	1,312	1,276	0,014	2,709	0,528	-0,023
JLS - LOCO-I	247	201	1,352	1,339	0,031	2,756	0,573	-0,175
HDP - LW	458	520	1,536	1,570	0,124	2,748	0,468	-0,103
EXR - PIZ	103	209	1,577	1,600	0,069	2,846	0,557	-0,276
EXR - ZIP	150	1094	2,180	2,278	0,043	4,464	1,006	-0,108



*Obr.56 Graf výsledku testů pro obrázky v 8 bitové hloubce*

Nejlepší kompresí v této kategorii je PNG-DEFLATE\_9. Kompresní poměr vykazuje ve všech případech nejlepší hodnoty. Jen o asi 2 % horší je komprese PNG-DEFLATE\_1, která svou rychlostí komprese překonává PNG-DEFLATE\_9 téměř 6-krát.

Dvě výše zmíněné komprese byly jediné, které nedosáhly záporné komprese. Naopak u novějších typů formátů JPEG 2000, JPEG-LS, HD Photo, EXR-PIZ a EXR-ZIP se záporné komprese objevují nejčastěji. U těchto formátů k tomu však spíše dochází kvůli nutnosti převodu na větší bitovou hloubku než kvůli špatnému kompresnímu algoritmu. Tyto formáty by neměly být využívány pro barevné obrázky v 8 bitové hloubce.

Příloha P XII zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 1152×870 (1 002 204 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.49*.

*Tab.49 Výsledky testů pro vybraný obrázek v 8 bitové hloubce*

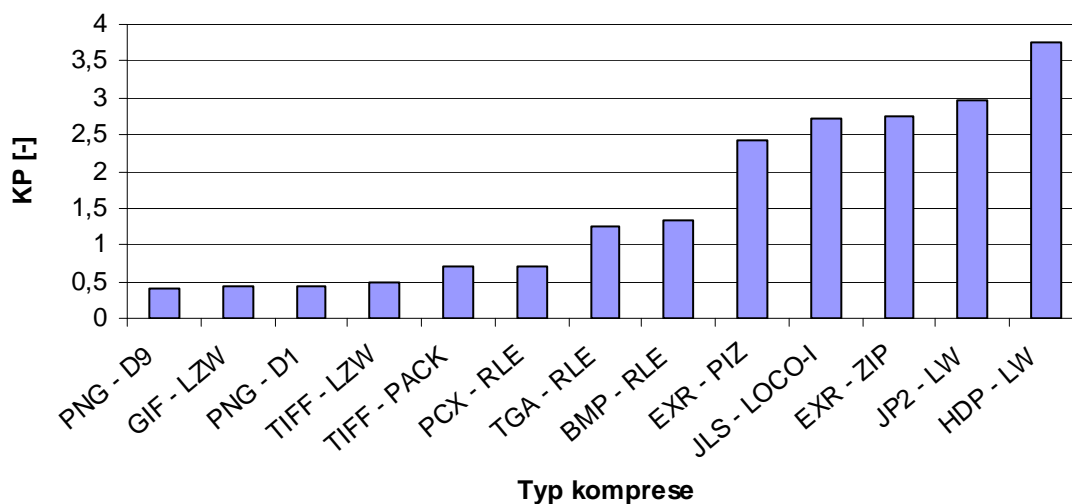
Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
PNG - D9	13	337	0,522	0,142
PNG - D1	14	53	0,541	0,866
GIF - LZW	85	124	0,593	0,328
TIFF - LZW	13	38	0,602	1,048
TIFF - PACK	9	29	0,792	0,716
TGA - RLE	14	33	0,834	0,504
PCX - RLE	27	11	0,804	1,780
BMP - RLE	25	12	0,843	1,306
JP2 - LW	898	1480	1,655	-0,044
JLS - LOCO-I	223	186	1,536	-0,288
HDP - LW	403	451	1,741	-0,164
EXR - PIZ	100	200	1,752	-0,376
EXR - ZIP	133	1043	2,353	-0,130

### 6.2.3 Obrázky ve 4 bitové hloubce

Celkem bylo testováno 635 obrázků s průměrným počtem pixelů 774 558. Výsledky testů jsou uvedeny v Tab.50 a grafické zobrazení průměru kompresních poměrů je na Obr.57.

Tab.50 Výsledky testů pro obrázky ve 4 bitové hloubce

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
PNG - D9	5	293	0,407	0,416	0,013	1,153	0,179	0,202
GIF - LZW	27	33	0,431	0,450	0,027	1,015	0,178	1,734
PNG - D1	6	13	0,441	0,450	0,024	1,153	0,175	4,233
TIFF - LZW	5	23	0,487	0,502	0,044	2,831	0,247	2,234
TIFF - PACK	4	13	0,695	0,739	0,104	2,723	0,255	2,396
PCX - RLE	10	4	0,700	0,735	0,109	1,226	0,255	8,221
TGA - RLE	8	19	1,240	1,310	0,105	6,385	0,537	-1,257
BMP - RLE	20	11	1,322	1,395	0,106	8,538	0,640	-2,985
EXR - PIZ	65	128	2,409	2,608	0,233	6,550	0,852	-1,103
JLS - LOCO-I	114	91	2,722	2,881	0,126	5,498	1,037	-1,895
EXR - ZIP	86	448	2,760	2,717	0,086	5,811	1,330	-0,393
JP2 - LW	594	984	2,969	3,161	0,203	5,993	1,081	-0,200
HDP - LW	285	326	3,746	3,947	0,452	10,324	1,058	-0,843



Obr.57 Graf výsledku testů pro obrázky ve 4 bitové hloubce

V kategorii 4 bitových obrázků je z hlediska průměru kompresních poměrů, mediánu a minimálního kompresního poměru nejlepší komprese PNG-DEFLATE\_9. Problémy jsou však s vyšší hodnotou doby komprese, která výrazně převyšuje ostatní hodnoty v popředí tabulky.

V této kategorii se nepodařilo najít typ komprese, která by někdy nedosahovala záporných hodnot. Nejlépe je v tomto ohledu formát GIF, dosáhl nejnižšího maximálního kompresního poměru. Pokud se k tomu připočte druhý nejlepší kompresní poměr a asi 9-krát rychlejší komprese než u formátu PNG-DEFLATE\_9, jedná se o nejlepší volbu právě v případě upřednostnění rychlosti komprese před nepatrně lepším kompresním poměrem. Další možnou volbou v tomto směru je i komprese PNG-DEFLATE\_1, která je ještě 3-krát rychlejší než GIF a horší jen asi o 1 % v průměru kompresních poměrů.

Příloha P XIII zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 1024×768 (786 432 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.51*.

*Tab.51 Výsledky testů pro vybraný obrázek ve 4 bitové hloubce*

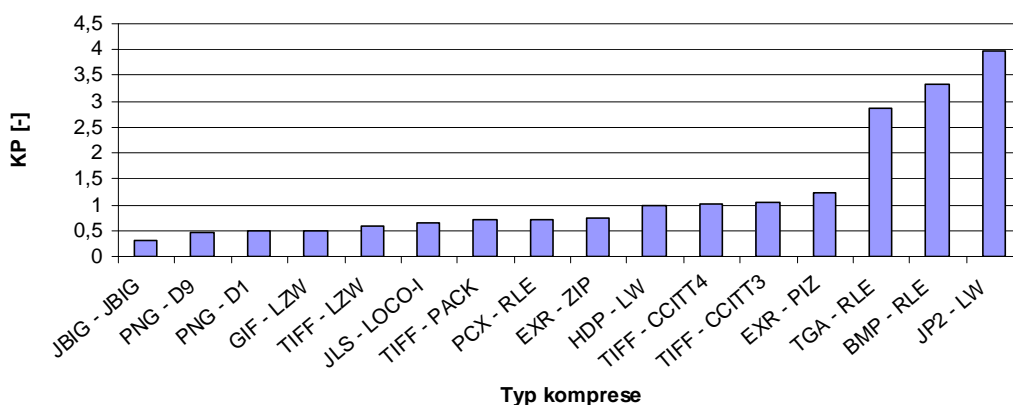
Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
PNG - D9	8	296	0,379	0,210
GIF - LZW	31	39	0,415	1,501
PNG - D1	1	23	0,441	2,430
TIFF - LZW	8	16	0,435	3,529
TIFF - PACK	16	8	0,782	2,723
PCX - RLE	16	1	0,731	26,938
TGA - RLE	16	16	1,475	-2,968
BMP - RLE	22	9	1,490	-5,440
EXR - PIZ	78	148	2,488	-1,005
JLS - LOCO-I	155	124	3,270	-1,830
EXR - ZIP	86	570	2,687	-0,296
JP2 - LW	663	1046	3,064	-0,197
HDP - LW	304	359	3,518	-0,701

### 6.2.4 Obrázky v 1 bitové hloubce

Celkem bylo testováno 1915 obrázků s průměrným počtem pixelů 940 427. Výsledky testů jsou uvedeny v *Tab.52* a grafické zobrazení průměru kompresních poměrů je na *Obr.58*.

*Tab.52 Výsledky testů pro obrázky v 1 bitové hloubce*

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
JBIG - JBIG	29	32	0,317	0,299	0,003	0,985	0,226	2,164
PNG - D9	3	77	0,452	0,435	0,008	1,945	0,281	0,710
PNG - D1	3	5	0,482	0,471	0,012	1,945	0,269	10,821
GIF - LZW	21	27	0,498	0,486	0,025	1,176	0,275	1,876
TIFF - LZW	3	19	0,583	0,521	0,043	3,405	0,402	2,246
JLS - LOCO-I	21	20	0,653	0,543	0,015	1,741	0,456	1,760
TIFF - PACK	2	10	0,705	0,675	0,051	3,243	0,372	2,851
PCX - RLE	3	2	0,717	0,721	0,058	1,811	0,324	18,214
EXR - ZIP	34	100	0,745	0,712	0,046	4,284	0,479	0,255
HDP - LW	86	97	0,978	0,881	0,031	3,063	0,595	0,023
TIFF - CCITT4	7	22	1,011	0,675	0,006	3,529	0,929	-0,051
TIFF - CCITT3	6	20	1,038	0,770	0,018	3,486	0,899	-0,189
EXR - PIZ	42	72	1,230	1,069	0,122	6,892	0,911	-0,320
TGA - RLE	9	21	2,881	2,024	0,137	11,216	2,507	-8,991
BMP - RLE	21	10	3,340	3,015	0,083	15,000	2,419	-23,288
JP2 - LW	266	444	3,985	3,592	0,090	7,998	2,471	-0,673



*Obr.58 Graf výsledku testů pro obrázky v 1 bitové hloubce*

V této kategorii je jasně nejlepším řešením použití komprese JBIG. Je to jediný případ, jehož kompresní poměr byl při testech vždy pod hodnotou 1. I když z pohledu rychlosti komprese představuje JBIG spíše průměr, příliš se nevyplatí upřednostňovat kompresi PNG-DEFLATE\_1. Ta má sice asi 6-krát rychlejší kompresní algoritmus, ovšem kompresní poměr se pohybuje v hodnotách horších asi o 17 %, což nelze zanedbat.

Milým překvapením jsou formáty JPEG-LS a EXR-ZIP, které i přes nutnost zvýšení bitové hloubky dosáhly v průměru nezáporné komprese. JPEG-LS měla dokonce třetí nejlepší maximální hodnotu kompresního poměru.

Naopak komprese RLE ve formátech TGA a BMP i vlnkové transformace ve formátech JPEG 2000 a EXR dokazují, že při použití těchto typů kompresí si není možné dovolit zvyšování bitové hloubky.

Nemilým překvapením je komprese TIFF-CCITT3 a TIFF-CCITT4, které dosahují v průměru záporné komprese. Je ale nutné přihlídnout k tomu, že tyto komprese byly vytvořeny převážně pro textový obsah v obrázku a navíc pro medián kompresních poměrů dosahují výrazně lepších hodnot.

U obrázků, kterým bylo sníženo počet barev pomocí Floyd a Steinbergovi metody bylo dosaženo horších výsledků kompresních poměrů než u obrázků původně vytvořených ve dvou barvách.

Příloha P XIV zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 959×875 (839 125 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.53*.

Tab.53 Výsledky testů pro vybraný obrázek v 1 bitové hloubce

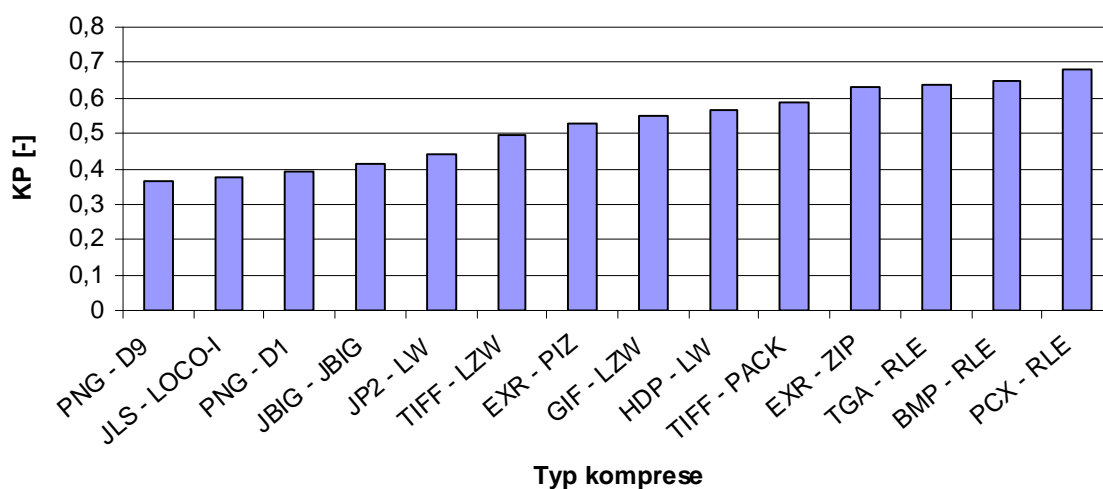
Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
JBIG - JBIG	15	24	0,338	2,757
PNG - D9	1	86	0,411	0,685
PNG - D1	8	1	0,453	54,679
GIF - LZW	16	24	0,470	2,206
TIFF - LZW	1	24	0,491	2,122
JLS - LOCO-I	14	14	0,586	2,955
TIFF - PACK	1	16	0,538	2,889
PCX - RLE	8	1	0,579	42,094
EXR - ZIP	31	94	0,648	0,375
HDP - LW	62	78	0,789	0,270
TIFF - CCITT4	15	16	0,787	1,329
TIFF - CCITT3	1	39	0,803	0,506
EXR - PIZ	47	78	0,823	0,227
TGA - RLE	1	24	2,305	-5,438
BMP - RLE	17	10	2,336	-13,363
JP2 - LW	250	406	3,585	-0,637

### 6.2.5 Obrázky v odstínech šedi

Celkem bylo testováno 436 obrázků s průměrným počtem pixelů 3 614 476. Výsledky testů jsou uvedeny v Tab.54 a grafické zobrazení průměru kompresních poměrů je na Obr.59.

Tab.54 Výsledky testů pro obrázky v odstínech šedi

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
PNG - D9	59	1441	0,367	0,249	0,004	0,932	0,254	0,044
JLS - LOCO-I	167	148	0,376	0,356	0,014	0,929	0,220	0,421
PNG - D1	61	223	0,392	0,272	0,009	0,935	0,255	0,272
JBIG - JBIG	971	915	0,413	0,348	0,006	1,036	0,265	0,064
JP2 - LW	787	1291	0,440	0,447	0,014	0,957	0,211	0,043
TIFF - LZW	46	96	0,495	0,352	0,035	1,330	0,347	0,526
EXR - PIZ	117	221	0,529	0,579	0,039	1,114	0,248	0,213
GIF - LZW	195	283	0,548	0,439	0,016	1,332	0,372	0,160
HDP - LW	446	498	0,565	0,580	0,115	0,981	0,198	0,087
TIFF - PACK	14	40	0,588	0,489	0,032	1,008	0,354	1,030
EXR - ZIP	138	755	0,633	0,469	0,023	1,654	0,419	0,049
TGA - RLE	27	83	0,634	0,611	0,032	1,071	0,361	0,439
BMP - RLE	33	25	0,648	0,690	0,027	1,125	0,345	1,401
PCX - RLE	72	25	0,682	0,580	0,047	1,816	0,405	1,276



Obr.59 Graf výsledku testů pro obrázky v odstínech šedi

Nejlepší kompresí pro obrázky v odstínech šedi je PNG-DEFLATE a JPEG-LS. Pro kompresi DEFLATE\_9 a DEFLATE\_1 hovoří také medián kompresních poměrů, které dosahují nejlepších výsledků. Na druhou stranu komprese JPEG-LS je rychlejší a dosáhla nejlepšího maximálního kompresního poměru. Navíc nejsou v průměru kompresních poměrů výrazné rozdíly.

Záporných kompresí nedosahovaly formáty PNG-DEFLATE\_9, JPEG-LS, PNG-DEFLATE\_1, JPEG 2000 a HD Photo. Poslední jmenovaný formát však kvůli horšímu průměru kompresních poměrů není pro tento typ obrázků doporučen.

Příloha P XV zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 2481×1747 (4 334 307 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.55*.

*Tab.55 Výsledky testů pro vybraný obrázek v odstínech šedi*

Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
PNG - D9	87	2987	0,335	0,022
JLS - LOCO-I	223	193	0,329	0,348
PNG - D1	94	299	0,382	0,207
JBIG - JBIG	1259	1226	0,339	0,054
JP2 - LW	1295	2206	0,436	0,026
TIFF - LZW	63	131	0,455	0,416
EXR - PIZ	161	296	0,464	0,181
GIF - LZW	334	409	0,483	0,126
HDP - LW	634	690	0,590	0,059
TIFF - PACK	25	71	0,561	0,618
EXR - ZIP	176	913	0,588	0,045
TGA - RLE	38	114	0,587	0,362
BMP - RLE	89	33	0,601	1,208
PCX - RLE	92	32	0,571	1,339

## 6.3 Text

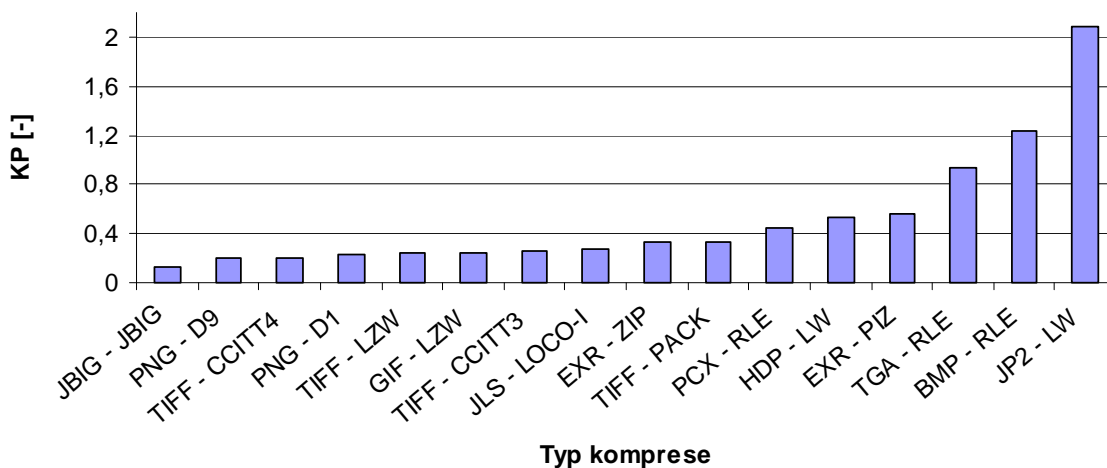
Obrázky pro tyto testy obsahovaly textovou informaci, někdy i v kombinaci s obrázkem.

### 6.3.1 Text v 1 bitové hloubce

Celkem bylo testováno 357 obrázků s průměrným počtem pixelů 2 861 160. Výsledky testů jsou uvedeny v *Tab.56* a grafické zobrazení průměru kompresních poměrů je na *Obr.60*.

Tab.56 Výsledky testů pro obrázky s textem v 1 bitové hloubce

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
JBIG - JBIG	58	67	0,126	0,129	0,017	0,552	0,054	1,306
PNG - D9	5	147	0,197	0,199	0,032	0,627	0,072	0,545
TIFF - CCITT4	7	20	0,198	0,189	0,024	1,219	0,112	4,024
PNG - D1	4	9	0,228	0,228	0,041	0,662	0,076	8,945
TIFF - LZW	4	21	0,247	0,247	0,057	0,792	0,085	3,549
GIF - LZW	45	63	0,249	0,250	0,044	0,723	0,084	1,198
TIFF - CCITT3	7	19	0,257	0,257	0,052	1,324	0,123	3,864
JLS - LOCO-I	25	23	0,277	0,277	0,043	0,933	0,104	3,203
EXR - ZIP	94	211	0,328	0,327	0,082	1,002	0,110	0,319
TIFF - PACK	2	11	0,334	0,344	0,076	0,740	0,103	5,919
PCX - RLE	6	2	0,440	0,454	0,097	0,946	0,139	23,229
HDP - LW	220	248	0,538	0,541	0,085	1,707	0,199	0,186
EXR - PIZ	103	177	0,560	0,562	0,182	1,486	0,173	0,248
TGA - RLE	20	55	0,929	0,914	0,211	3,845	0,396	0,129
BMP - RLE	37	15	1,232	1,251	0,166	4,167	0,525	-1,573
JP2 - LW	617	1080	2,083	2,103	0,276	6,846	0,776	-0,100



Obr.60 Graf výsledku testů pro obrázky s textem v 1 bitové hloubce

Pro tuto kategorii je nejlepší využít kompresi JBIG. Tato komprese dosahuje nejlepšího průměru, mediánu, minimálního i maximálního kompresního poměru. Při požadavku na rychlost komprese je vhodnější využití komprese TIFF-CCITT4, která je rychlejší více než 3-krát. Na druhou stranu JBIG nedosahuje záporných kompresí a kompresní poměr je lepší

asi o 7 %. Horší komprese však dosahovaly formáty u subjektů s textem a obrázkem - u čistého textu k záporným kompresím u TIFF-CCITT4, TIFF-CCITT3 apod. nedocházelo.

Další možnou variantou je při preferování rychlosti komprese formát PNG-DEFLATE\_1, který je ještě 2-krát rychlejší než TIFF-CCITT4 a nedosáhl záporné komprese. Průměrná hodnota kompresních poměrů je o asi 3 % horší než u TIFF-CCITT4.

Formáty, u kterých se neprojevila záporná komprese jsou JBIG, PNG-DEFLATE\_9, PNG-DEFLATE\_1, TIFF-LZW, GIF-LZW, JPEG-LS, TIFF-PACKBITS, PCX. Kvůli horším hodnotám průměru kompresních poměrů formátů JPEG-LS, TIF-PACKBITS a PCX však není doporučeno příliš tyto formáty využívat. Naopak jak již bylo zmíněno výše, lepší variantou je komprese TIFF-CCITT4, a to i přesto, že záporné komprese dosáhla.

Příloha P XVI zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 1383×2144 (2 965 152 pixelů). Výsledky tohoto obrázku ukazuje *Tab.57*.

*Tab.57 Výsledky testů pro vybraný obrázek s textem v 1 bitové hloubce*

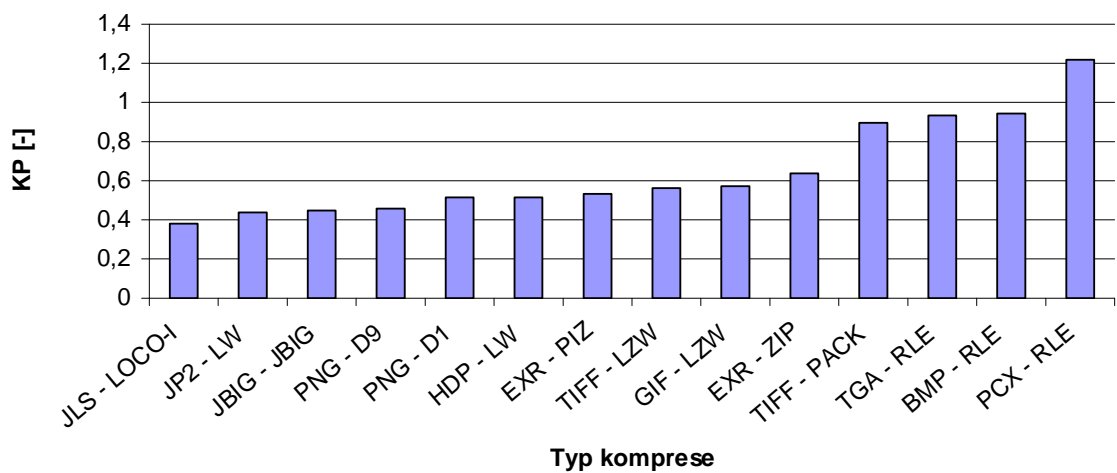
Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
JBIG - JBIG	62	63	0,108	1,416
PNG - D9	1	187	0,224	0,415
TIFF - CCITT4	16	16	0,156	5,273
PNG - D1	16	1	0,263	73,676
TIFF - LZW	1	16	0,276	4,528
GIF - LZW	55	70	0,282	1,026
TIFF - CCITT3	16	16	0,198	5,014
JLS - LOCO-I	30	30	0,302	2,325
EXR - ZIP	94	218	0,351	0,298
TIFF - PACK	1	8	0,440	6,996
PCX - RLE	8	1	0,559	44,122
HDP - LW	219	258	0,595	0,157
EXR - PIZ	117	195	0,595	0,208
TGA - RLE	15	63	0,805	0,309
BMP - RLE	38	15	0,850	1,000
JP2 - LW	749	1326	2,323	-0,100

### 6.3.2 Text v odstínech šedi

Celkem bylo testováno 319 obrázků s průměrným počtem pixelů 2 854 891. Výsledky testů jsou uvedeny v *Tab.58* a grafické zobrazení průměru kompresních poměrů je na *Obr.61*.

*Tab.58 Výsledky testů pro obrázky s textem v odstínech šedi*

Typ K	DD [ms]	DK [ms]	KP <sub>φ</sub>	KP <sub>med</sub>	KP <sub>min</sub>	KP <sub>max</sub>	σ <sub>KP</sub>	ČEK [%/ms]
JLS - LOCO-I	212	190	0,377	0,365	0,140	0,623	0,081	0,328
JP2 - LW	783	1269	0,433	0,429	0,176	0,622	0,078	0,045
JBIG - JBIG	865	833	0,449	0,442	0,146	0,716	0,092	0,066
PNG - D9	67	3034	0,459	0,448	0,173	0,697	0,085	0,018
PNG - D1	72	249	0,512	0,506	0,200	0,718	0,076	0,196
HDP - LW	405	444	0,514	0,512	0,281	0,702	0,070	0,109
EXR - PIZ	134	232	0,534	0,537	0,220	0,735	0,085	0,201
TIFF - LZW	47	90	0,561	0,552	0,205	0,934	0,121	0,486
GIF - LZW	206	304	0,567	0,555	0,171	0,931	0,123	0,142
EXR - ZIP	115	672	0,638	0,626	0,283	1,027	0,139	0,054
TIFF - PACK	17	39	0,891	0,881	0,284	1,005	0,074	0,277
TGA - RLE	27	77	0,930	0,916	0,298	1,060	0,081	0,091
BMP - RLE	46	27	0,945	0,945	0,303	1,007	0,061	0,204
PCX - RLE	70	28	1,219	1,155	0,398	1,751	0,199	-0,769



*Obr.61 Graf výsledku testů pro obrázky s textem v odstínech šedi*

V tomto testu výrazně převyšuje komprese LOCO-I formátu JPEG-LS všechny kompresní algoritmy, a to i z hlediska rychlosti komprese. U maximálního kompresního poměru byl sice lepší formát JPEG 2000, ovšem jen o 0,1 %, což je velmi nepatrný rozdíl.

Záporných kompresí dosáhly formáty EXR-ZIP, TIFF-PACKBITS, TGA, BMP a PCX a i z pohledu na hodnoty průměrů kompresních poměrů těchto formátů je zřejmé, že pro obrázky testované v této kategorii není jejich použití vhodné.

Příloha P XVII zobrazuje průměrný testovaný obrázek v této kategorii. Původní rozlišení obrázku bylo 1197×1546 (1 850 562 pixelů). Výsledky testu tohoto obrázku ukazuje *Tab.59*.

*Tab.59 Výsledky testů pro vybraný obrázek s textem v odstínech šedi*

Typ K	DD [ms]	DK [ms]	KP	ČEK [%/ms]
JLS - LOCO-I	132	116	0,363	0,549
JP2 - LW	499	796	0,429	0,072
JBIG - JBIG	522	515	0,442	0,108
PNG - D9	39	3222	0,448	0,017
PNG - D1	47	148	0,506	0,334
HDP - LW	257	274	0,514	0,177
EXR - PIZ	102	172	0,541	0,267
TIFF - LZW	31	55	0,557	0,806
GIF - LZW	125	179	0,575	0,237
EXR - ZIP	78	476	0,637	0,076
TIFF - PACK	16	24	0,892	0,448
TGA - RLE	15	55	0,928	0,130
BMP - RLE	47	16	0,954	0,287
PCX - RLE	47	23	1,181	-0,785

## ZÁVĚR

Cílem této diplomové práce bylo vytvořit literární rešerši na téma neztrátové kompresní algoritmy v rastrové počítačové grafice. Dalším bodem bylo seznámit se s formáty, které tyto algoritmy využívají a popsat jejich strukturu. Následovalo naprogramování aplikace s implementací neztrátových algoritmů a popis této aplikace. Nakonec byly tyto algoritmy testovány a vyhodnoceny na dvanácti různých kolekcích obrázků.

Bylo zjištěno, že u fotografií a 24 bitových obrázků je nejlepší volbou využití formátu JPEG 2000 s vlnkovou transformací. Tento formát disponuje výborným kompresním poměrem, jehož hodnoty jsou často nejlepší ze všech testovaných formátů. Jedinou chybou formátu JPEG 2000 je doba komprese, která v testech patří k jedné z nejdelších.

Pokud je dán požadavek na rychlou a zároveň kvalitní kompresi, tak je nejvhodnější použít kompresní algoritmus LOCO-I formátu JPEG-LS. I když tato komprese dosahuje u barevných fotografií a 24 bitových obrázků dobrých výsledků, kvalita této komprese se projevuje hlavně u obrázků v odstínech šedi - bez ohledu na to, zda se jedná o fotografie, obrázky vytvořené v počítači nebo obsahující text.

Přestože komprese JBIG podporuje obrázky v odstínech šedi, nejedná se o prioritu této komprese. Skvělý kompresní poměr dosahuje u obrázků v 1 bitové hloubce - bez ohledu na to, zda jde čistě o obrázky, o obrázky s textem nebo něco mezi.

Výsledky komprese CCITT4 a CCITT3 potvrdily, že jejich primárním využitím jsou 1 bitové obrázky s textem, u kterých obzvlášť komprese CCITT4 dosahovala velmi dobrých hodnot. Pokud se však v obrázku nevyskytuje text, tak často dochází k záporné kompresi. Rychlost komprese je u těchto dvou algoritmů totožná.

U 8 a 4 bitových obrázků je nejlepší možností využití komprese DEFLATE formátu PNG. V úrovních komprese DEFLATE není příliš velký rozdíl, a je tedy na volbě uživatele jakou vybere. Formát PNG je nejuniverzálnějším formátem - ve všech testovaných kategoriích patří kompresní algoritmus tohoto formátu k jednomu z nejlepších z hlediska kompresního poměru, a při snížení úrovně komprese i z hlediska rychlosti komprese.

U fotografií s vysokou dynamikou barev ukazuje svou sílu formát EXR s kompresí PIZ a ZIP. Z pohledu kompresního poměru není mezi těmito dvěma typy komprese větší rozdíl. S přihlédnutím k rychlosti komprese je však PIZ lepší volbou.

Komprese LZW u formátů GIF a TIFF je vhodné použít jen u obrázků s bitovou hloubkou nižší než 8 bitů. U ostatních obrázků dosahuje komprese LZW spíše průměrných hodnot. Obecně se dá říct, že je lepší upřednostňovat kompresi LZW formátu GIF, která dokáže o něco více snížit velikost souboru než LZW formátu TIFF.

Převážně komprese formátu JPEG 2000, ale i HD Photo, EXR a JPEG-LS se nehodí při použití na barevné obrázky s bitovou hloubkou 8 a nižší.

Komprese RLE formátů PCX, TGA, BMP a komprese PACKBITS formátu TIFF dosahují z dnešního pohledu špatných kompresních poměrů ve všech kategoriích a nevyplatí se je tedy příliš využívat.

## ZÁVĚR V ANGLIČTINĚ

The aim of this diploma thesis was to create survey describing about lossless compression algorithms in raster computer graphics. The next item was to become familiar with formats, that use these algorithms and to describe structure of these formats. It is followed by programming of an application with lossless algorithms implementation and by description of the application. At the end, these algorithms were tested and evaluated on twelve different image collections.

It was found, the best choice for photographs and 24 bit depth pictures is JPEG 2000 format with wavelet transformation. This format has an excellent compression ratio, whose values are often the best of all tested formats. The only flaw is the JPEG 2000 compression time, which in tests belong to one of the longest.

If the request is made for both quality and fast compression, so it is the best to use a LOCO-I compression of JPEG-LS format. Although this compression reaches good results for photographs and 24 bit depth images, quality of this compression is manifested mainly in greyscale images - irrespective of whether they are photographs, images created in a computer or images containing the text.

Although JBIG compression supports greyscale images, these are not its priority. The excellent compression ratio is reached with 1 bit depth images - irrespective of whether it is purely images, the images with text or something between these two choices.

CCITT4 and CCITT3 compression results confirmed, that their primary use are 1 bit images with text. Specially CCITT4 compression make very good progress. However, if the text is not in the image, it often produces negative compression. Compression times are identical for these two algorithms.

For 8 and 4 bit images is the best possible to use DEFLATE compression of PNG format. The DEFLATE compression levels is not much difference, so a user themself can make decision about choosing the compression level. PNG is the most universal format of all - the compression algorithm of this format belongs to one of the best in view of compression ratio and of compression time (valid only for small compression level) in all tested categories.

EXR format including PIZ and ZIP compressions shows its strength for photos with high dynamic colours. In view of compression ratio, between these two types of compression are not a big differences. The PIZ is a little bit better than ZIP because of quicker compression.

LZW compression in GIF and TIFF formats should be used only for images with the bit depth less than 8 bits. LZW compression reaches average values of compression ratio for other types of images. Generally we can say, that it is better to prefer GIF-LZW compression, which can reduce the file size more than TIFF-LZW.

Mostly JPEG 2000, as well as HD Photo, EXR, JPEG-LS is not suitable for use on colour images with the bit depth of 8 or less.

RLE compression of PCX, TGA, BMP and PACKBITS compression of TIFF achieve poor compression ratio in all categories and therefore they are not recommended to use.

**SEZNAM POUŽITÉ LITERATURY**

- [1] ŽÁRA, Jiří, Bedřich BENEŠ, Jiří SOCHOR a Petr FELKEL. *Moderní počítačová grafika: Kompletní průvodce metodami 2D a 3D grafiky*. 2. vydání. Brno: Computer Press, 2004. ISBN 80-251-0454-0.
- [2] STRACHOTA, Pavel. *Ukládání a komprese obrazu* [online]. Praha: FJFI ČVUT, 2010, 15.9.2010 [cit. 2012-01-08]. Dostupné z: [http://saint-paul.fjfi.cvut.cz/base/public-filesystem/admin-upload/POGR/POGR1/07.ukladani\\_a\\_kompresse\\_obrazu.pdf](http://saint-paul.fjfi.cvut.cz/base/public-filesystem/admin-upload/POGR/POGR1/07.ukladani_a_kompresse_obrazu.pdf)
- [3] MURRAY, James D. a William VANRYPER. *Encyklopedie grafických formátů*. 2. vydání. Praha: Computer Press, 1997. ISBN 80-7226-033-2.
- [4] PELIKÁN, Josef. *Kódování rastrových obrázků* [online]. Praha: CGG MFF UK, 2011, 16 s. [cit. 2012-03-23]. Dostupné z: <http://cgg.mff.cuni.cz/~pepca/lectures/pdf/pg1-19-imagecoding.pdf>
- [5] MORKEŠ, David. *Komprimační a archivační programy*. Praha: Computer Press, 1998. ISBN 80-7226-089-8.
- [6] VEČERKA, Arnošt. *Kompresse dat* [online]. Olomouc: Univerzita Palackého, 2008, 30.4.2008 [cit. 2011-12-18]. Dostupné z: <http://phoenix.inf.upol.cz/esf/ucebni/kompresse.pdf>
- [7] GIMLI. *Kompresse a kompresní algoritmy. Gimliho stránky* [online]. [2006-2010] [cit. 2012-01-09]. Dostupné z: <http://gimli.mysteria.cz/kompresse/algoritmy.html>
- [8] SALOMON, David. *Data Compression: the complete reference*. 4th edition. London: Springer, 2007, 1092 s. ISBN 1-84628-602-6.
- [9] MATOUŠEK, Radomil. *Metody kódování* [online]. verze 1.9. Brno: VUT, 2006, 96 s. [cit. 2012-03-26]. Dostupné z: <http://www.uai.fme.vutbr.cz/~matousek/TIK/TIKv9.swf>
- [10] KUHN, Markus. *JBIG1 patent information. The Computer Laboratory: Faculty of Computer Science and Technology* [online]. 26.2.2011 [cit. 2012-04-30]. Dostupné z: <http://www.cl.cam.ac.uk/~mgk25/jbigkit/patents/>

- [11] HATLAPATKA, Radim. *JBIG2 komprese* [online]. Brno, 2009 [cit. 2012-03-11]. Dostupné z: <http://www.fi.muni.cz/~xhatlap/bc/bakalarka.pdf>. Bakalářská práce. Masarykova univerzita, Fakulta informatiky. Vedoucí práce doc. RNDr. Petr Sojka, Ph.D.
- [12] JBIG2 Development. JOINT PHOTOGRAPHIC EXPERTS GROUP. *The JPEG committee home page* [online]. 2007 [cit. 2012-03-11]. Dostupné z: <http://www.jpeg.org/jbig/jbigpt2.html>
- [13] ŠMÍD, Radislav. *Úvod do vlnkové transformace* [online]. Praha: ČVUT FEL, 9.8.2001, 9 s. [cit. 2012-03-25]. Dostupné z: <http://measure.feld.cvut.cz/usr/staff/smid/wavelets/Wavelet-intro8859.pdf>
- [14] VRÁNA, Jaroslav, Jan ČERMÁK a Radek ZEZULA. Výpočet vlnkové transformace pomocí algoritmu "lifting". *Elektrorevue* [online]. Ústav telekomunikací FEKT, 8.6.2004 [cit. 2012-03-25]. Dostupné z: <http://www.elektrorevue.cz/clanky/04034/index.html>
- [15] VRÁNA, Jaroslav. Využití adaptivního liftingu při kompresi dat. *Elektrorevue* [online]. Ústav telekomunikací FEKT, 10.2.2006 [cit. 2012-03-25]. Dostupné z: <http://www.elektrorevue.cz/clanky/06010/index.html>
- [16] *DZS - cvičení č. 6 - DPCM, DCT a její použití při kompresi obrazové informace* [online]. 23.5.2007, 5 s. [cit. 2012-03-24]. Dostupné z: [www.comtel.cz/files/download.php?id=3119](http://www.comtel.cz/files/download.php?id=3119)
- [17] BALÍK, Miroslav, Marek HANUŠ, Jan HOLUB a Michal PAULÍČEK. PPMC. *Knihovna vizualizačních appletů pro kompresi dat* [online]. 2006 [cit. 2012-03-24]. Dostupné z: [http://www.stringology.org/DataCompression/ppmc/index\\_cs.html](http://www.stringology.org/DataCompression/ppmc/index_cs.html)
- [18] TIŠNOVSKÝ, Pavel. Grafický formát BMP - používaný a přitom neoblíbený. *Root.cz* [online]. Internet Info, 19.10.2006 [cit. 2012-04-02]. Dostupné z: <http://www.root.cz/clanky/graficky-format-bmp-pouzivany-a-pritom-neoblíbeny/>
- [19] TIŠNOVSKÝ, Pavel. BMP na vícero způsobů. *Root.cz* [online]. Internet Info, 26.10.2006 [cit. 2012-04-02]. Dostupné z: <http://www.root.cz/clanky/bmp-na-vicero-zpusobu/>

- [20] TIŠNOVSKÝ, Pavel. Případ GIF. *Root.cz* [online]. Internet Info, 10.8.2006 [cit. 2012-03-03]. Dostupné z: <http://www.root.cz/clanky/pripad-gif/>
- [21] TIŠNOVSKÝ, Pavel. Pravda a mýty o GIFu. *Root.cz* [online]. Internet Info, 17.8.2006 [cit. 2012-03-03]. Dostupné z: <http://www.root.cz/clanky/pravda-a-myty-o-gifu/>
- [22] COMPUSERVE. *GRAPHICS INTERCHANGE FORMAT. Version 89a* [online]. 1990 [cit. 2012-03-05]. Dostupné z: <http://www.w3.org/Graphics/GIF/spec-gif89a.txt>
- [23] TIŠNOVSKÝ, Pavel. Anatomie grafického formátu GIF. *Root.cz* [online]. Internet Info, 24.8.2006 [cit. 2012-03-05]. Dostupné z: <http://www.root.cz/clanky/anatomie-grafickeho-formatu-gif/>
- [24] TIŠNOVSKÝ, Pavel. GIF: animace a konkurence. *Root.cz* [online]. Internet Info, 31.8.2006 [cit. 2012-03-05]. Dostupné z: <http://www.root.cz/clanky/gif-animace-a-konkurence/>
- [25] TIŠNOVSKÝ, Pavel. Grafický formát PCX - výlet do historie PC. *Root.cz* [online]. Internet Info, 16.11.2006 [cit. 2012-02-26]. Dostupné z: <http://www.root.cz/clanky/graficky-format-pcx-vylet-do-historie-pc/>
- [26] TIŠNOVSKÝ, Pavel. PCX prakticky - implementace komprimace RLE. *Root.cz* [online]. Internet Info, 23.11.2006 [cit. 2012-02-28]. Dostupné z: <http://www.root.cz/clanky/pcx-prakticky-implementace-komprimace-rle/>
- [27] TIŠNOVSKÝ, Pavel. PNG is Not GIF. *Root.cz* [online]. Internet Info, 7.9.2006 [cit. 2012-03-08]. Dostupné z: <http://www.root.cz/clanky/png-is-not-gif/>
- [28] Portable Network Graphics (PNG) Specification (Second Edition): Information technology — Computer graphics and image processing — Portable Network Graphics (PNG): Functional specification. ISO/IEC 15948:2003 (E). W3C. *World Wide Web Consortium (W3C)* [online]. 10.11.2003 [cit. 2012-03-09]. Dostupné z: <http://www.w3.org/TR/PNG/>
- [29] TIŠNOVSKÝ, Pavel. Řádkové filtry v PNG. *Root.cz* [online]. Internet Info, 29.9.2006 [cit. 2012-03-08]. Dostupné z: <http://www.root.cz/clanky/radkove-filtry-v-png/>

- [30] TIŠNOVSKÝ, Pavel. Anatomie grafického formátu PNG. *Root.cz* [online]. Internet Info, 14.9.2006 [cit. 2012-03-08]. Dostupné z: <http://www.root.cz/clanky/anatomie-grafickeho-formatu-png/>
- [31] TIŠNOVSKÝ, Pavel. PNG - bity, byty, chunky. *Root.cz* [online]. Internet Info, 21.9.2006 [cit. 2012-03-08]. Dostupné z: <http://www.root.cz/clanky/png-bity-byty-chunky/>
- [32] TIŠNOVSKÝ, Pavel. Grafický formát TGA - jednoduchý, oblíbený, používaný. *Root.cz* [online]. Internet Info, 2.11.2006 [cit. 2012-02-28]. Dostupné z: <http://www.root.cz/clanky/graficky-format-tga-jednoduchy-oblibeny-pouzivany/>
- [33] TRUEVISION. *Truevision TGA: FILE FORMAT SPECIFICATION 2.0* [online]. 1991 [cit. 2012-02-28]. Dostupné z: <http://www.dca.fee.unicamp.br/~martino/disciplinas/ea978/tgaffs.pdf>
- [34] TIŠNOVSKÝ, Pavel. Grafický formát TGA prakticky. *Root.cz* [online]. Internet Info, 9.11.2006 [cit. 2012-02-28]. Dostupné z: <http://www.root.cz/clanky/graficky-format-tga-prakticky/>
- [35] BOURKE, Paul. Creating TGA Image files. *Paul Bourke* [online]. Zář 1996 [cit. 2012-03-04]. Dostupné z: <http://paulbourke.net/dataformats/tga/>
- [36] Truevision TGA. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001-2012 [cit. 2012-03-04]. Dostupné z: [http://en.wikipedia.org/wiki/Truevision\\_TGA](http://en.wikipedia.org/wiki/Truevision_TGA)
- [37] WARMERDAM, Frank, Andrey KISELEV, Mike WELLES a Dwight KELLY. Modifying The TIFF Library. *LibTIFF - TIFF Library and Utilities* [online]. 1997, 23.12.2003 [cit. 2012-03-14]. Dostupné z: <http://www.libtiff.org/internals.html>
- [38] ADOBE DEVELOPERS ASSOCIATION. *TIFF: Revision 6.0* [online]. 3.6.1992 [cit. 2012-03-14]. Dostupné z: <http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf>
- [39] KVÁŠ, Marek. Kompresce JPEG 2000 a akcelerace pomocí DSP. *Elektrorevue* [online]. 9.4.2008, roč. 2008, č. 13, s. 12 [cit. 2012-03-15]. ISSN 1213-1539.

- Dostupné z: <http://www.elektrorevue.cz/cz/download/kompresa-jpeg-2000-akcelerace-pomoci-dsp/>
- [40] ISO/IEC JTC1/SC29 WG1. *INFORMATION TECHNOLOGY: JPEG 2000 IMAGE CODING SYSTEM* [online]. Martin Boliek. 16.3.2000, 11.4.2000 [cit. 2012-03-16]. Dostupné z: <http://www.jpeg.org/public/fcd15444-1.pdf>
- [41] MICROSOFT. *HD Photo: Photographic Still Image File Format* [online]. 16.11.2006, 43 s. [cit. 2012-03-18]. Dostupné z: [http://download.microsoft.com/download/7/4/3/74394b57-2028-4859-a668-c4c0af0726e2/HDPhoto\\_Feature\\_Spec\\_1.0.doc](http://download.microsoft.com/download/7/4/3/74394b57-2028-4859-a668-c4c0af0726e2/HDPhoto_Feature_Spec_1.0.doc)
- [42] ITU-T. *T.832: Information technology – JPEG XR image coding system – Image coding specification* [online]. Březen 2009, 212 s. [cit. 2012-03-18]. Dostupné z: [http://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-T.832-200903-S!!PDF-E&type=items](http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.832-200903-S!!PDF-E&type=items)
- [43] WEINBERGER, Marcelo, Gadiel SEROUSSI a Guillermo SAPIRO. HEWLETT-PACKARD. *The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS* [online]. 9. vydání. USA, Srpen 2000, 34 s. [cit. 2012-03-22]. Dostupné z: <http://www.hpl.hp.com/loco/HPL-98-193R1.pdf>
- [44] ITU-T. *T.87: Information technology – Lossless and near-lossless compression of continuous-tone still images – Baseline* [online]. 18.6.1998, 75 s. [cit. 2012-03-22]. Dostupné z: [http://www.itu.int/rec/dologin\\_pub.asp?lang=e&id=T-REC-T.87-199806-I!!ZPF-E&type=items](http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-T.87-199806-I!!ZPF-E&type=items)
- [45] JPEG image compression FAQ, part 1/2: Section - [13] Isn't there a lossless JPEG?. ADVAMEG. *Internet FAQ Archives* [online]. 28.3.1999, 21.11.2011 [cit. 2012-03-22]. Dostupné z: <http://www.faqs.org/faqs/jpeg-faq/part1/section-13.html>
- [46] WALLACE, Gregory K. *The JPEG Still Picture Compression Standard* [online]. Prosinec 1991, 17 s. [cit. 2012-03-22]. Dostupné z: <http://white.stanford.edu/~brian/psy221/reader/Wallace.JPEG.pdf>
- [47] OpenEXR: nový open-source formát pro počítačovou grafiku. KREJČÍ, Richard. *Grafika* [online]. 12.2.2003 [cit. 2012-03-17]. Dostupné z: <http://www.grafika.cz/art/dv/openexr.html>

- [48] KAINZ, Florian a Rod BOGART. INDUSTRIAL LIGHT & MAGIC. *Technical Introduction to OpenEXR* [online]. 18.2.2009, 15 s. [cit. 2012-03-17]. Dostupné z: <http://www.openexr.com/TechnicalIntroduction.pdf>
- [49] INDUSTRIAL LIGHT & MAGIC. *OpenEXR File Layout* [online]. 24.4.2007, 11 s. [cit. 2012-03-17]. Dostupné z: <http://www.openexr.com/openexrfilelayout.pdf>
- [50] MASSIMINO, Pascal. Experimental branch. In: *Groups Google* [online]. 28.11.2011 [cit. 2012-03-22]. Dostupné z: [https://groups.google.com/a/webmproject.org/group/webp-discuss/browse\\_thread/thread/bf368050925aeb8e#](https://groups.google.com/a/webmproject.org/group/webp-discuss/browse_thread/thread/bf368050925aeb8e#)
- [51] Lossless and Transparency Encoding in WebP. GOOGLE. *WebP* [online]. 17.11.2011 [cit. 2012-03-19]. Dostupné z: [http://code.google.com/intl/cs/speed/webp/docs/webp\\_lossless\\_alpha\\_study.html](http://code.google.com/intl/cs/speed/webp/docs/webp_lossless_alpha_study.html)
- [52] Webp lossless – first impressions. *I am an extreme moderate* [online]. 20.11.2011 [cit. 2012-03-19]. Dostupné z: <http://extrememoderate.wordpress.com/2011/11/20/webp-lossless-first-impressions/>
- [53] RATUSHNYAK, Alexander. FLIC - a new fast lossless image compressor. In: *Encode's Forum* [online]. Listopad 2011 [cit. 2012-03-19]. Dostupné z: <http://encode.ru/threads/1222-FLIC-a-new-fast-lossless-image-compressor>
- [54] A web-centric image compression benchmark. *I am an extreme moderate* [online]. 28.11.2011 [cit. 2012-03-19]. Dostupné z: <http://extrememoderate.wordpress.com/2011/11/28/a-web-centric-image-compression-benchmark/>
- [55] IFRAH, ERAN. *CodeLite with MinGW and wxWidgets* [software]. Ver. 2.10.0.4778, Ver. 4.4.1, Ver. 2.8.12. 12.4.2011. [cit. 2012-04-27]. Dostupné z: <http://sourceforge.net/projects/codelite/files/Releases/codelite-2.10/codelite-2.10.0.4778-mingw4.4.1-wx2.8.12.exe/download>
- [56] HURTADO, Jose Antonio, RJP COMPUTING, RJMYST3, Michal BLIŽŇÁK, Jérémie FOUCHÉ. *wxFormBuilder* [software]. Ver. 3.3.0-beta. 2.12.2011 [cit. 2012-04-27]. Dostupné z:

[http://sourceforge.net/projects/wxformbuilder/files/wxformbuilder-nightly/3.3.0-beta/wxFormBuilder\\_v3.3.0-beta.exe/download](http://sourceforge.net/projects/wxformbuilder/files/wxformbuilder-nightly/3.3.0-beta/wxFormBuilder_v3.3.0-beta.exe/download)

- [57] DROLON Hervé. *FreeImage* [software]. Ver. 3.15.3. 17.3.2012 [cit. 2012-04-27]. Dostupné z: <http://downloads.sourceforge.net/freeimage/FreeImage3153Win32.zip>
- [58] KUHN Markus. *JBIG-KIT* [software]. Ver. 2.0. 30.8.2008 [cit. 2012-04-30]. Dostupné z: <http://www.cl.cam.ac.uk/~mgk25/download/jbigkit-2.0.tar.gz>
- [59] MICROSOFT. *HD Photo Device Porting Kit 1.0* [software]. Ver. 1.0. 3.4.2010 [cit. 2012-04-27]. Dostupné z: [http://download.microsoft.com/download/1/7/4/1743e193-c99b-4ffa-9365-b3bb4c2a736a/hdphoto\\_1.0\\_dpk\\_setup.exe](http://download.microsoft.com/download/1/7/4/1743e193-c99b-4ffa-9365-b3bb4c2a736a/hdphoto_1.0_dpk_setup.exe)
- [60] HEWLETT-PACKARD. *Loco executables* [software]. Ver. 1.0.0. Říjen 1999 [cit. 2012-04-27]. Dostupné z: <http://www.hpl.hp.com/loco/jlsrefV100.zip>
- [61] DROLON, Hervé. *FreeImage: a free, open source graphics library* [online]. 3.15.3. 17.3.2012, 129 s. [cit. 2012-04-27]. Dostupné z: <http://downloads.sourceforge.net/freeimage/FreeImage3153.pdf>
- [62] *Ulož.to* [online]. Nodus Technologies, 2012 [cit. 2012-05-11]. Dostupné z: <http://www.uloz.to/>
- [63] *Photojournal: NASA's Image Access Home Page* [online]. JPL, 2012 [cit. 2012-05-11]. Dostupné z: <http://photojournal.jpl.nasa.gov/index.html>
- [64] The New Test Images. *Image Compression* [online]. 2011 [cit. 2012-05-11]. Dostupné z: [http://www.imagecompression.info/test\\_images/](http://www.imagecompression.info/test_images/)
- [65] *PhotographyBlog* [online]. 2012 [cit. 2012-05-11]. Dostupné z: <http://www.photographyblog.com/>
- [66] Recognition Benchmark Images. *The University of Kentucky Center for Visualization & Virtual Environments* [online]. 2009 [cit. 2012-05-11]. Dostupné z: <http://vis.uky.edu/~stewe/ukbench/>

- [67] The USC-SIPI Image Database. *USC Ming Hsieh Department of Electrical Engineering - Signal and Image Processing Institute* [online]. 2012 [cit. 2012-05-11]. Dostupné z: <http://sipi.usc.edu/database/database.php>
- [68] Light Probe Image Gallery. DEBEVEC, Paul. *Paul Debevec Home Page* [online]. 2010 [cit. 2012-05-12]. Dostupné z: <http://www.pauldebevec.com/Probes/>
- [69] OpenEXR Downloads. *OpenEXR* [online]. Industrial Light and Magic, 2012 [cit. 2012-05-12]. Dostupné z: <http://www.openexr.com/downloads.html>

**SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK**

2D	2 Dimenze - dvojrozměrný objekt
ASCII	American Standard Code for Information Interchange - typ kódování textové informace
B	Byte - jednotka množství dat v informatice.
BGR	barevný prostor RGB uložený v pořadí barev ve formátu B, G a R
BH	bitová hloubka obrázku
BMQ	NuCore Raw Image File - nezpracované obrazové data firmy NuCore
BMP	Microsoft Windows Bitmap - grafický formát pro ukládání rastrové počítačové grafiky.
BMP - RLE	formát BMP s kompresí RLE
BP	barevný prostor obrázku
bpB	bits per Byte - počet bitů na 1 Byte. Jednotka pro vyjádření jednoho z možných typů kompresního poměru.
CCITT	International Telegraph and Telephone Consultative Committee - standardizační organizace (nyní ITU-T)
CD	Compact Disc - optický disk určený pro ukládání digitálních dat
CMYK	barevný prostor založený na subtraktivním míchání barev
CR	ASCII znak Cursor Return s hodnotou 0Dh
CR2	Canon Digital Camera RAW Image Format version 2.0 - nezpracované obrazové data firmy Canon. Základem tohoto formátu je struktura TIFF
CSV	Comma-Separated Values (hodnoty oddělené čárkami) - souborový formát určený pro výměnu tabulkových dat
ČEK	Časová Efektivita Komprese
DD	Doba Dekomprese algoritmu

DK	Doba Komprese algoritmu
DNG	Adobe Digital Negative - nezpracované obrazové data firmy Adobe
DPCM	Differential Pulse Code Modulation - rozdílová pulsně kódová modulace
DWT	Discrete Wavelet Transformation - diskrétní vlnková transformace
EBCOT	Embedded Block Coding with Optimal Truncation - metoda aritmetického entropického kódování
EXR - PIZ	grafický formát EXR s kompresí PIZ
EXR - ZIP	grafický formát EXR s kompresí ZIP
FCT	Forward Core Transform - hlavní dopředná transformace
FIR	Finite Impulse Response - filtr s konečnou impulzní odezvou
G31D	CCITT Group 3 jednoúrovňové
G32D	CCITT Group 3 dvourozměrné
G42D	CCITT Group 4 dvourozměrné
GIF	Graphics Interchange Format - grafický formát pro ukládání rastrové počítačové grafiky.
GIF - LZW	formát GIF s kompresí LZW
GUI	Graphical User Interface - grafické uživatelské rozhraní
HD Photo	High Definition Photo - grafický formát firmy Microsoft, speciálně navržený pro fotografie (známý také jako Windows Media Photo)
HDP - LW	formát HD Photo s kompresí Lossless Wavelet
HDR	High Dynamic Range - vysoký dynamický rozsah (barev)
ICC	International Colour Consortium
ID	identifikační kód
IDAT	datová část (pixmap) formátu PNG

IEND	ukončující značka formátu PNG
IFD	Image File Directory - adresář souboru předlohy formátu TIFF
IFH	Image File Header - hlavička souboru předlohy formátu TIFF
IHDR	hlavička obrázku formátu PNG
ITU-T	International Telecommunication Union - Telecommunication Standardization Sector - mezinárodní telekomunikační unie (dříve CCITT)
IWT	Integer Wavelet Transformation - Celočíslná vlnková transformace
JBIG	Joint Bi-level Image Experts Group - neztrátová komprese obrazu, popř. formát, který využívá kompresi JBIG
JBIG - JBIG	formát JBIG s kompresí JBIG
JLS - LOCO-I	formát JPEG-LS s kompresí LOCO-I
JP2 - LW	formát JPEG 2000 s kompresí Lossless Wavelet
JPEG	Joint Photographic Experts Group - grafický formát se ztrátovou kompresí
JPEG 2000	Joint Photographic Experts Group 2000 - grafický formát se ztrátovou a neztrátovou kompresí
JPEG-LS	Joint Photographic Experts Group - Lossless - grafický formát s neztrátovou a mírně ztrátovou kompresí
kB	kiloByte - $2^{10}$ Byte.
KP	kompresní poměr algoritmu
$KP_{\phi}$	průměr kompresních poměrů
$KP_{med}$	medián kompresních poměrů
$KP_{min}$	minimální kompresní poměr
$KP_{max}$	maximální kompresní poměr
LF	ASCII znak Line Feed s hodnotou 0Ah

---

LJPEG	Lossless Joint Photographic Experts Group - původní implementace JPEG s neztrátovou kompresí
LOCO-I	LOW COMplexity LOSSless COMpression for Images - typ neztrátové komprese používaný ve formátu JPEG-LS
LZ77	kompresní slovníková metoda vytvořena Lempem a Zivem v roce 1977
LZSS	Lempel-Ziv-Storer-Szymanski - modifikace kompresní metody LZ77
LZW	Lempel-Ziv-Welch - neztrátový slovníkový kompresní algoritmus
MB	MegaByte - $2^{20}$ Byte.
NEF	Nikon Digital Camera Raw Image Format - nezpracované obrazové data firmy Nikon
OpenEXR	grafický formát pro obrázky s vysokou dynamikou barev
ORF	Olympus Digital Camera Raw Image Format - nezpracované obrazové data firmy Olympus
OŠ	odstíny šedi
PC	Personal Computer - osobní počítač
PCX	Personal Computer eXchange - formát pro ukládání rastrové počítačové grafiky.
PCX - RLE	grafický formát PCX s kompresí RLE
PFM	Portable Floatmap - formát s podporou barevného prostoru RGBF
PNG	Portable Network Graphics - grafický formát určený pro beztrátovou kompresi rastrové grafiky.
PNG - D1	formát PNG s kompresí Deflate úrovně 1
PNG - D9	formát PNG s kompresí Deflate úrovně 9
PPM	Prediction by Partial Matching - adaptivní statistická technika komprese
RAF	Fuji Digital Camera Raw Image Format - nezpracované obrazové data

	firmy Fuji
RAM	Random-Access Memory - operační paměť s přímým přístupem
RAW	označení obrázkových souborů obsahující nezpracovaná data
RGB	barevný prostor s barvami Red (červená), Green (zelená) a Blue (modrá)
RGBA	barevný prostor RGB s alfa kanálem (průhlednost)
RGBA	barevný prostor RGBF s alfa kanálem (průhlednost)
RGBF	barevný prostor RGB. Každá složka je uložena datovým typem Float.
RLE	Run-Length Encoding - komprese proudového kódování
TGA	Truevision Graphics Adapter - formát pro ukládání rastrové počítačové grafiky.
TGA - RLE	formát TGA s kompresí RLE
TIFF	Tag Image File Format - formát pro ukládání rastrové počítačové grafiky.
TIFF - DEFL	formát TIFF s kompresí DEFLATE
TIFF - CCITT3	formát TIFF s kompresí CCITT3
TIFF - CCITT4	formát TIFF s kompresí CCITT4
TIFF - LZW	formát TIFF s kompresí LZW
TIFF - PACK	formát TIFF s kompresí PACKBITS
Typ K	typ komprese
YUV	barevný model používaný v televizním vysílání v normě PAL
WWW	World Wide Web - celosvětové propojení počítačových sítí
$\sigma_{KP}$	směrodatná odchylka kompresních poměrů

## SEZNAM OBRÁZKŮ

<i>Obr.1 Geometrická reprezentace prostoru RGB [1]</i> .....	13
<i>Obr.2 Geometrická reprezentace prostoru CMY [1]</i> .....	15
<i>Obr.3 Geometrická reprezentace prostoru HSV (vlevo) a HLS (vpravo) [1]</i> .....	17
<i>Obr.4 Fyzické a logické pixely [3]</i> .....	18
<i>Obr.5 Princip použití indexového módu pro určení barev [3]</i> .....	20
<i>Obr.6 Princip použití direct módu pro určení barev [1]</i> .....	20
<i>Obr.7 Typy palet [3]</i> .....	21
<i>Obr.8 Schéma pixelového zhušťování [3]</i> .....	27
<i>Obr.9 Základní schéma algoritmu RLE [5]</i> .....	29
<i>Obr.10 Varianty proudového kódování [3]</i> .....	30
<i>Obr.11 RLE paket na bitové úrovni [3]</i> .....	31
<i>Obr.12 RLE paket na Bytové úrovni [1]</i> .....	32
<i>Obr.13 RLE paket na pixelové úrovni [3]</i> .....	32
<i>Obr.14 RLE paket se třemi Byty [3]</i> .....	33
<i>Obr.15 RLE schémata s vertikálními replikačními pakety [3]</i> .....	35
<i>Obr.16 Příklad postupu algoritmu LZW při kódování [5]</i> .....	39
<i>Obr.17 Příklad binárního stromu Huffmanova kódování [5]</i> .....	41
<i>Obr.18 Kódování CCITT G31D [3]</i> .....	45
<i>Obr.19 Příklad binárního stromu Shannon-Fanova kódování [5]</i> .....	46
<i>Obr.20 Časově-kmitočtové rozlišení vlnkové transformace [13]</i> .....	54
<i>Obr.21 Frekvenční pohled na diskrétní vlnkovou transformaci [13]</i> .....	56
<i>Obr.22 Jeden krok DWT a rozklad na aproximace a detaily [13]</i> .....	56
<i>Obr.23 Blokové schéma několikasupňového celočíselného liftingu [15]</i> .....	58
<i>Obr.24 Blokové schéma predikčního kodéru a dekodéru [16]</i> .....	59
<i>Obr.25 Princip dvourozměrné predikce [16]</i> .....	59
<i>Obr.26 Postup komprese PPMC [6]</i> .....	62
<i>Obr.27 Vytvoření rotace bloku textu [6]</i> .....	63
<i>Obr.28 Logická obrazovka a rámce formátu GIF [21]</i> .....	69
<i>Obr.29 Prokládaný režim formátu GIF</i> .....	71
<i>Obr.30 Prokládaný režim formátu PNG</i> .....	81
<i>Obr.31 Tři typy uspořádání dat formátu TIFF [3]</i> .....	96

<i>Obr.32 Logická organizace souboru TIFF [3]</i> .....	97
<i>Obr.33 Kódový stream formátu JPEG 2000 [40]</i> .....	100
<i>Obr.34 Postup komprese formátu JPEG 2000 [39]</i> .....	101
<i>Obr.35 Postup komprese formátu HD Photo</i> .....	104
<i>Obr.36 Pozice sousedních hodnot využívaných predikcí ve formátu JPEG-LS [8]</i> .....	106
<i>Obr.37 Postup komprese formátu JPEG-LS [43]</i> .....	106
<i>Obr.38 Pozice sousedních hodnot využívaných predikcí ve formátu LJPEG [46]</i> .....	107
<i>Obr.39 Postup komprese formátu LJPEG [46]</i> .....	108
<i>Obr.40 Záložka v menu - Soubor</i> .....	115
<i>Obr.41 Záložka v menu - Obrázek</i> .....	115
<i>Obr.42 Záložka v menu - Nápověda</i> .....	115
<i>Obr.43 Prohlížení obrázků v aplikaci</i> .....	116
<i>Obr.44 Dialog s nastavením komprese formátů</i> .....	117
<i>Obr.45 Průběh provádění testů</i> .....	119
<i>Obr.46 Report ve vytvořené aplikaci</i> .....	120
<i>Obr.47 Výsledky testů otevřené v aplikaci Microsoft Excel</i> .....	120
<i>Obr.48 Dialog pro snížení počtu barev</i> .....	121
<i>Obr.49 Dialog pro mapování barev formátu s vysokým dynamickým rozsahem</i> .....	122
<i>Obr.50 Graf výsledku testů pro fotografie formátu JPEG</i> .....	131
<i>Obr.51 Graf výsledku testů pro fotografie formátu RAW</i> .....	133
<i>Obr.52 Graf výsledku testů pro fotografie v odstínech šedi</i> .....	135
<i>Obr.53 Graf výsledku testů pro fotografie s vysokou dynamikou barev</i> .....	137
<i>Obr.54 Graf výsledku testů pro fotografie s vysokou dynamikou barev v odstínech šedi</i> .....	139
<i>Obr.55 Graf výsledku testů pro obrázky ve 24 bitové hloubce</i> .....	140
<i>Obr.56 Graf výsledku testů pro obrázky v 8 bitové hloubce</i> .....	142
<i>Obr.57 Graf výsledku testů pro obrázky ve 4 bitové hloubce</i> .....	144
<i>Obr.58 Graf výsledku testů pro obrázky v 1 bitové hloubce</i> .....	146
<i>Obr.59 Graf výsledku testů pro obrázky v odstínech šedi</i> .....	149
<i>Obr.60 Graf výsledku testů pro obrázky s textem v 1 bitové hloubce</i> .....	151
<i>Obr.61 Graf výsledku testů pro obrázky s textem v odstínech šedi</i> .....	153

**SEZNAM TABULEK**

<i>Tab.1 Konverze zlomku na binární vyjádření [5]</i> .....	48
<i>Tab.2 Struktura formátu BMP [18]</i> .....	66
<i>Tab.3 Hlavička formátu BMP [18]</i> .....	66
<i>Tab.4 Informační hlavička formátu BMP verze 3 [18]</i> .....	67
<i>Tab.5 Hodnoty úniku formátu BMP [19]</i> .....	68
<i>Tab.6 Bloky formátu GIF [23]</i> .....	72
<i>Tab.7 Rozšiřující grafický blok formátu GIF [23]</i> .....	73
<i>Tab.8 Bitové vzorky konkrétního souboru formátu GIF [23]</i> .....	74
<i>Tab.9 Verze formátu PCX [25]</i> .....	76
<i>Tab.10 Hlavička formátu PCX [25]</i> .....	76
<i>Tab.11 Povolené kombinace hodnot počtu bitů na pixel a bitových rovin PCX [25]</i> .....	77
<i>Tab.12 Typy filtrů formátu PNG [28]</i> .....	80
<i>Tab.13 Hlavička souboru formátu PNG [30]</i> .....	82
<i>Tab.14 Datová část chunku IHDR formátu PNG [30]</i> .....	85
<i>Tab.15 Typ kódování barev formátu PNG [30]</i> .....	85
<i>Tab.16 Struktura formátu ZLIB [28]</i> .....	86
<i>Tab.17 Chunk IEND formátu PNG [30]</i> .....	86
<i>Tab.18 Významné nepovinné chunky formátu PNG [30]</i> .....	87
<i>Tab.19 Hlavička formátu TGA [32]</i> .....	88
<i>Tab.20 Povolené hodnoty v položce „typ obrázku“ formátu TGA [32]</i> .....	89
<i>Tab.21 Hlavička formátu TIFF [3]</i> .....	94
<i>Tab.22 IFD formátu TIFF [3]</i> .....	94
<i>Tab.23 Tag formátu TIFF [3]</i> .....	95
<i>Tab.24 Hodnoty tagu Compression dle specifikace [38]</i> .....	95
<i>Tab.25 Doplnující hodnoty tagu Compression dle dokumentace knihovny libtiff [37]</i> .....	96
<i>Tab.26 Povinné segmenty v hlavní hlavičce formátu JPEG 2000 [40]</i> .....	99
<i>Tab.27 Povinné segmenty v hlavičce dlaždic formátu JPEG 2000 [40]</i> .....	100
<i>Tab.28 Základní struktura formátu JPEG-LS [44]</i> .....	105
<i>Tab.29 Typy predikce formátu LJPEG [46]</i> .....	108
<i>Tab.30 Povinné atributy v hlavičce formátu OpenEXR [48] [49]</i> .....	109
<i>Tab.31 Kompresní schémata formátu OpenEXR [48] [49]</i> .....	111

<i>Tab.32 Seznam souborů nutných pro spuštění aplikace .....</i>	114
<i>Tab.33 Volitelné formáty a komprese pro provádění testů.....</i>	118
<i>Tab.34 Význam zkratk v reportu .....</i>	123
<i>Tab.35 Význam zkratk v tabulkách s výsledky .....</i>	130
<i>Tab.36 Výsledky testů pro fotografie formátu JPEG.....</i>	131
<i>Tab.37 Výsledky testů pro vybranou fotografii formátu JPEG .....</i>	132
<i>Tab.38 Výsledky testů pro fotografie formátů RAW .....</i>	133
<i>Tab.39 Výsledky testů pro vybranou fotografii formátu RAW.....</i>	134
<i>Tab.40 Výsledky testů pro fotografie v odstínech šedi.....</i>	135
<i>Tab.41 Výsledky testů pro vybranou fotografii v odstínech šedi .....</i>	136
<i>Tab.42 Výsledky testů pro fotografie s vysokou dynamikou barev .....</i>	137
<i>Tab.43 Výsledky testů pro vybranou fotografii s vysokou dynamikou barev.....</i>	138
<i>Tab.44 Výsledky testů pro fotografie s vysokou dynamikou barev v odstínech šedi.....</i>	138
<i>Tab.45 Výsledky testů pro vybranou fotografii s vysokou dynamikou barev v odstínech šedi .....</i>	139
<i>Tab.46 Výsledky testů pro obrázky ve 24 bitové hloubce .....</i>	140
<i>Tab.47 Výsledky testů pro vybraný obrázek ve 24 bitové hloubce.....</i>	141
<i>Tab.48 Výsledky testů pro obrázky v 8 bitové hloubce .....</i>	142
<i>Tab.49 Výsledky testů pro vybraný obrázek v 8 bitové hloubce .....</i>	143
<i>Tab.50 Výsledky testů pro obrázky ve 4 bitové hloubce .....</i>	144
<i>Tab.51 Výsledky testů pro vybraný obrázek ve 4 bitové hloubce.....</i>	145
<i>Tab.52 Výsledky testů pro obrázky v 1 bitové hloubce .....</i>	146
<i>Tab.53 Výsledky testů pro vybraný obrázek v 1 bitové hloubce .....</i>	148
<i>Tab.54 Výsledky testů pro obrázky v odstínech šedi.....</i>	149
<i>Tab.55 Výsledky testů pro vybraný obrázek v odstínech šedi.....</i>	150
<i>Tab.56 Výsledky testů pro obrázky s textem v 1 bitové hloubce .....</i>	151
<i>Tab.57 Výsledky testů pro vybraný obrázek s textem v 1 bitové hloubce .....</i>	152
<i>Tab.58 Výsledky testů pro obrázky s textem v odstínech šedi.....</i>	153
<i>Tab.59 Výsledky testů pro vybraný obrázek s textem v odstínech šedi.....</i>	154

## SEZNAM PŘÍLOH

Příloha P I: Pixelové vyjádření formátu HD Photo

Příloha P II: Ukázky nastavení datového a zobrazovacího okna formátu OpenEXR

Příloha P III: Diagram třídy Obrazek

Příloha P IV: Diagram třídy FreeImageRozhraniClass

Příloha P V: Diagram třídy MainFrame

Příloha P VI: Vybraný obrázek pro kategorii: Fotografie formátu JPEG

Příloha P VII: Vybraný obrázek pro kategorii: Fotografie formátu RAW

Příloha P VIII: Vybraný obrázek pro kategorii: Fotografie v odstínech šedi

Příloha P IX: Vybraný obrázek pro kategorii: Fotografie s vysokou dynamikou barev

Příloha P X: Vybraný obrázek pro kategorii: Fotografie s vysokou dynamikou barev  
v odstínech šedi

Příloha P XI: Vybraný obrázek pro kategorii: Obrázky ve 24 bitové hloubce

Příloha P XII: Vybraný obrázek pro kategorii: Obrázky v 8 bitové hloubce

Příloha P XIII: Vybraný obrázek pro kategorii: Obrázky ve 4 bitové hloubce

Příloha P XIV: Vybraný obrázek pro kategorii: Obrázky v 1 bitové hloubce

Příloha P XV: Vybraný obrázek pro kategorii: Obrázky v odstínech šedi

Příloha P XVI: Vybraný obrázek pro kategorii: Text v 1 bitové hloubce

Příloha P XVII: Vybraný obrázek pro kategorii: Text v odstínech šedi

Příloha P XVIII: Sumarizace výsledků testů

## PŘÍLOHA P I: PÍXELOVÉ VYJÁDŘENÍ FORMÁTU HD PHOTO

Pixelové vyjádření pro barevný prostor RGB a BGR

Formát pixelů	Počet kanálů	Bitů na kanál	Bitů na pixel	Datový typ
24bppRGB	3	8	24	UINT
24bppBGR	3	8	24	UINT
32bppBGR	3	8	24	UINT
48bppRGB	3	16	48	UINT
48bppRGBFixedPoint	3	16	48	SINT
48bppRGBHalf	3	16	48	Float
96bppRGBFixedPoint	3	32	96	SINT
128bppRGBFloat	3	32	128	Float
16bppBGR555	3	5	16	UINT
16bppBGR565	3	5, 6, 5	16	UINT
32bppBGR101010	3	10	32	UINT
32bppRGBE	3	16	32	Float

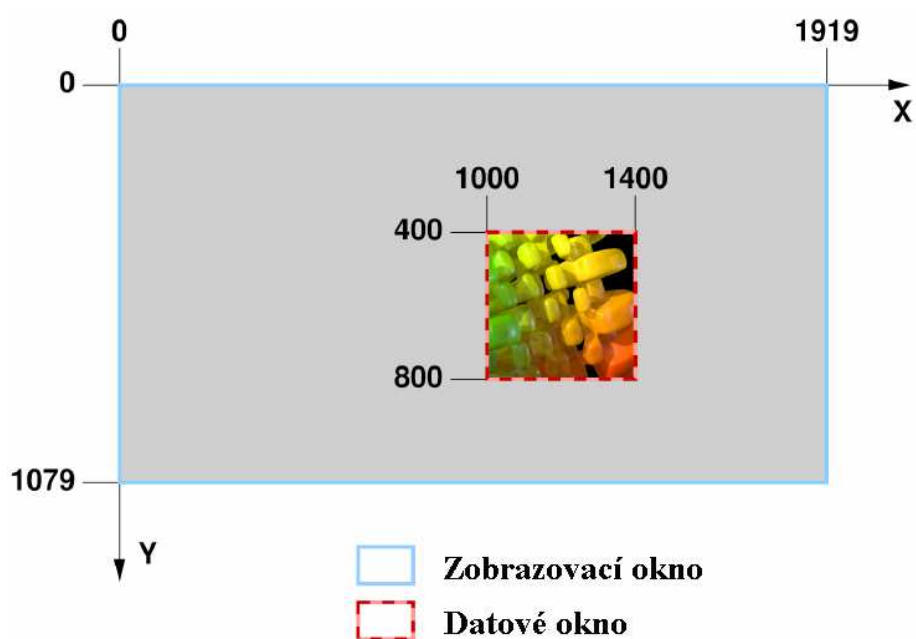
Pixelové vyjádření pro barevný prostor RGBA a BGRA

Formát pixelů	Počet kanálů	Bitů na kanál	Bitů na pixel	Datový typ
32bppBGRA	4	8	32	UINT
64bppRGBA	4	16	64	UINT
64bppRGBAFixedPoint	4	16	64	SINT
64bppRGBAHalf	4	16	64	Float
128bppRGBAFixedPoint	4	32	128	SINT
128bppRGBAFloat	4	32	128	Float

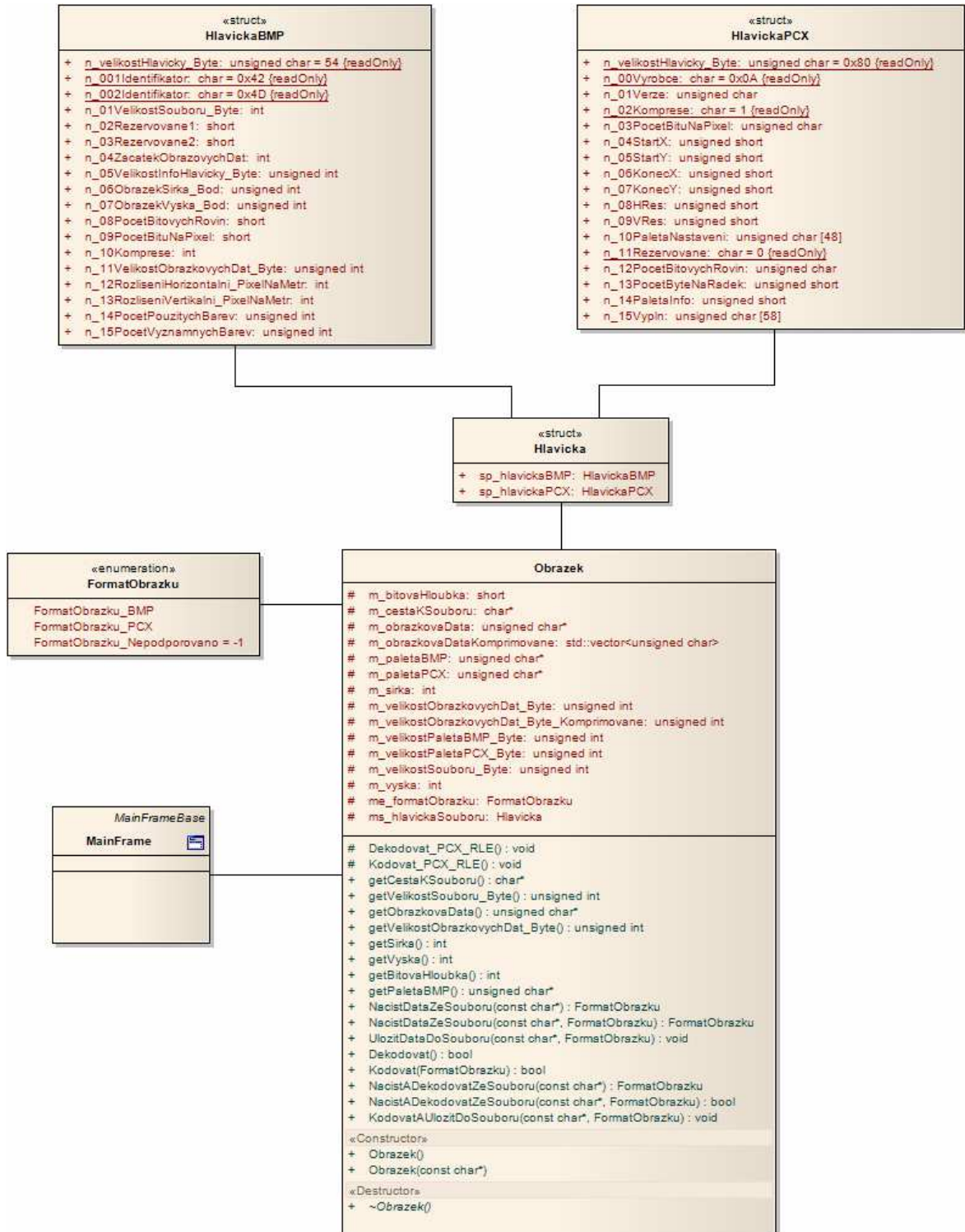
Pixelové vyjádření pro odstíny šedi

Formát pixelů	Počet kanálů	Bitů na kanál	Bitů na pixel	Datový typ
8bppGray	1	8	8	UINT
16bppGray	1	16	16	UINT
16bppGrayFixedPoint	1	16	16	SINT
16bppGrayHalf	1	16	16	Float
32bppGrayFixedPoint	1	32	32	SINT
32bppGrayFloat	1	32	32	Float
BlackWhite	1	1	1	UINT

## PŘÍLOHA P II: UKÁZKY NASTAVENÍ DATOVÉHO A ZOBRAZOVACÍHO OKNA FORMÁTU OPENEXR



# PŘÍLOHA P III: DIAGRAM TŘÍDY OBRAZEK



# PŘÍLOHA P IV: DIAGRAM TŘÍDY FREEIMAGEROZHRAICLASS



# PŘÍLOHA P V: DIAGRAM TŘÍDY MAINFRAME



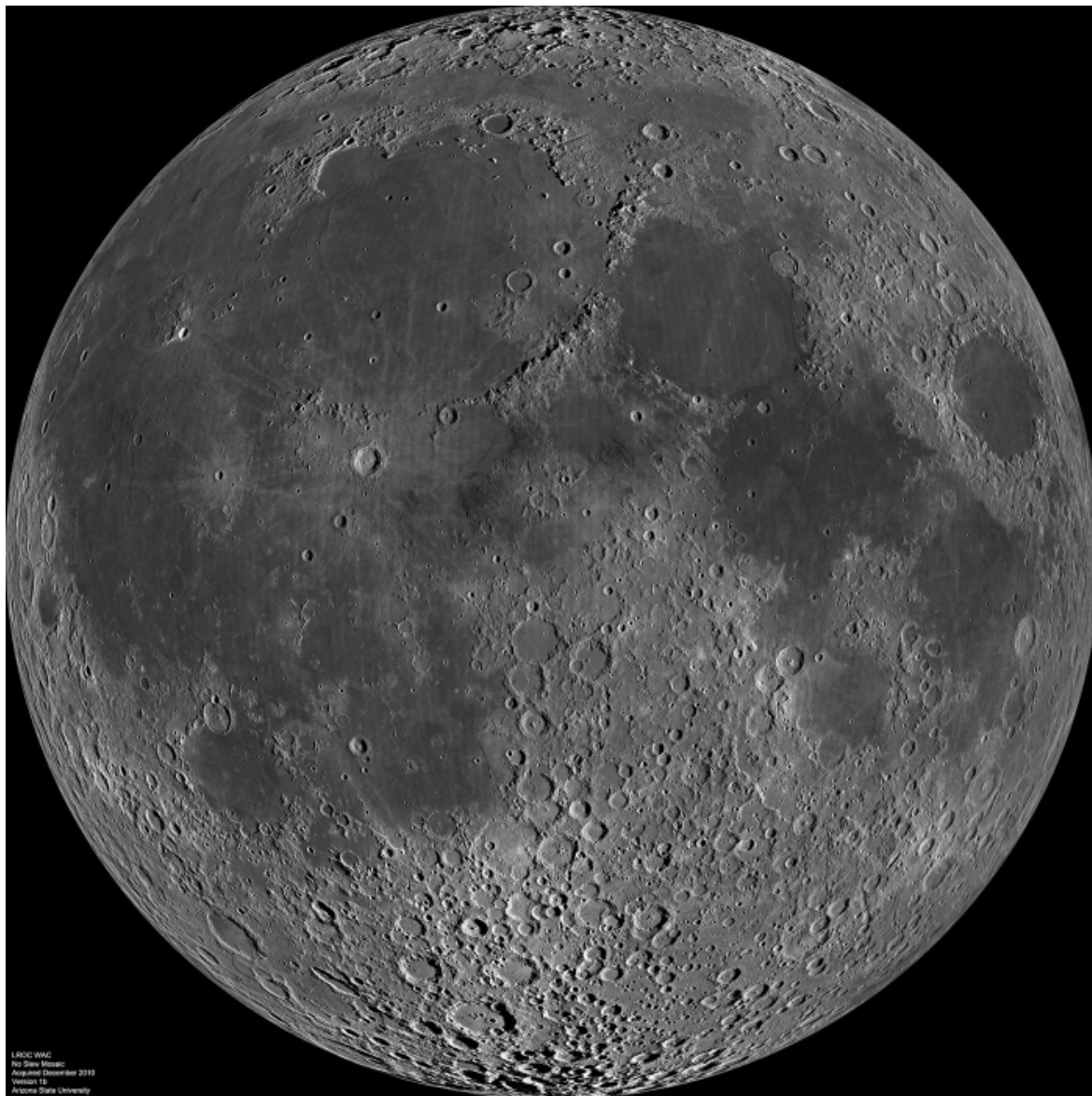
**PŘÍLOHA P VI: VYBRANÝ OBRÁZEK PRO KATEGORII:  
FOTOGRAFIE FORMÁTU JPEG**



**PŘÍLOHA P VII: VYBRANÝ OBRÁZEK PRO KATEGORII:  
FOTOGRAFIE FORMÁTU RAW**



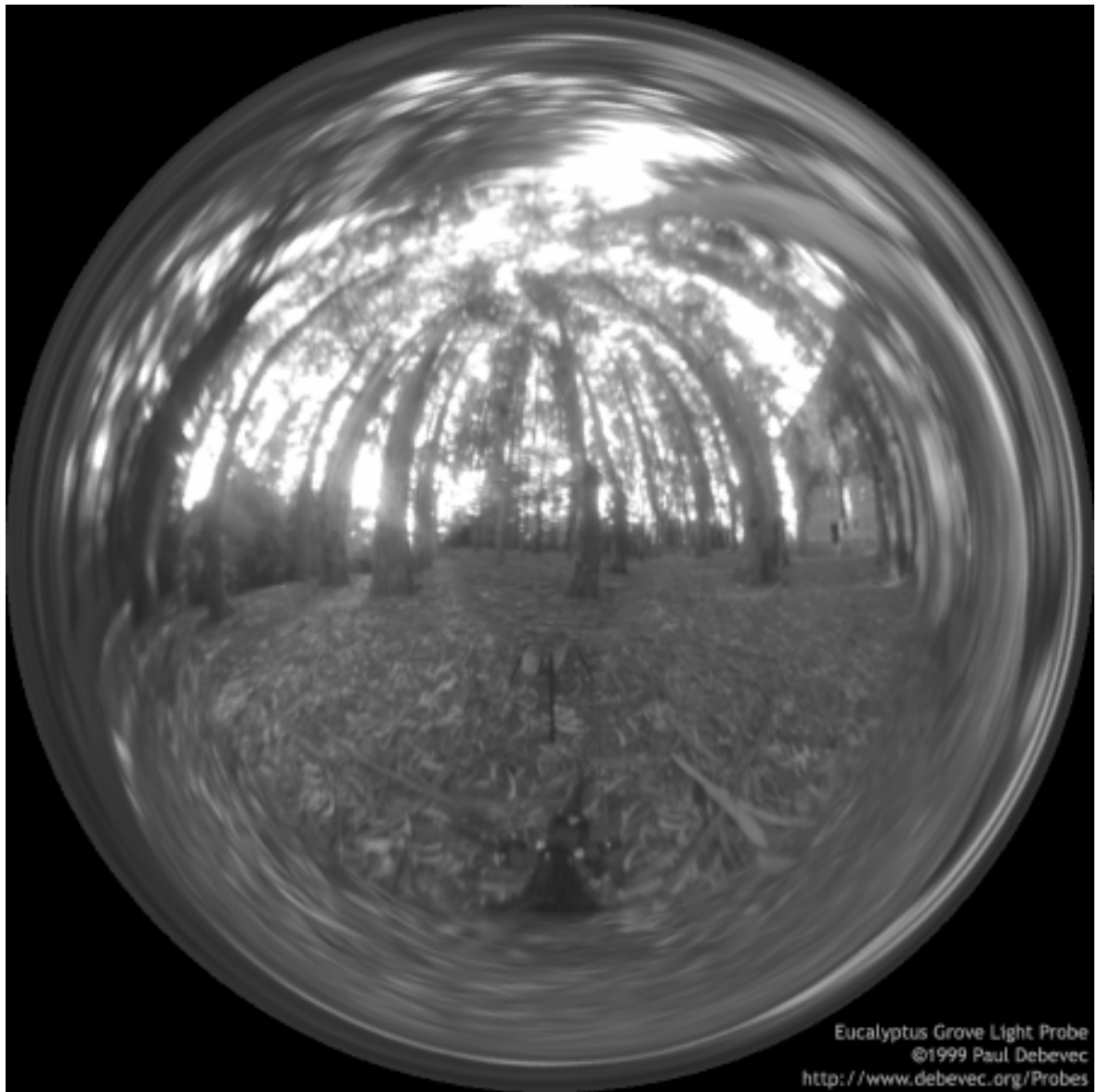
**PŘÍLOHA P VIII: VYBRANÝ OBRÁZEK PRO KATEGORII:  
FOTOGRAFIE V ODSTÍNECH ŠEDI**



**PŘÍLOHA P IX: VYBRANÝ OBRÁZEK PRO KATEGORII:  
FOTOGRAFIE S VYSOKOU DYNAMIKOU BAREV**



**PŘÍLOHA P X: VYBRANÝ OBRÁZEK PRO KATEGORII:  
FOTOGRAFIE S VYSOKOU DYNAMIKOU BAREV V ODSTÍNECH  
ŠEDI**



**PŘÍLOHA P XI: VYBRANÝ OBRÁZEK PRO KATEGORII:  
OBRÁZKY VE 24 BITOVÉ HLOUBCE**



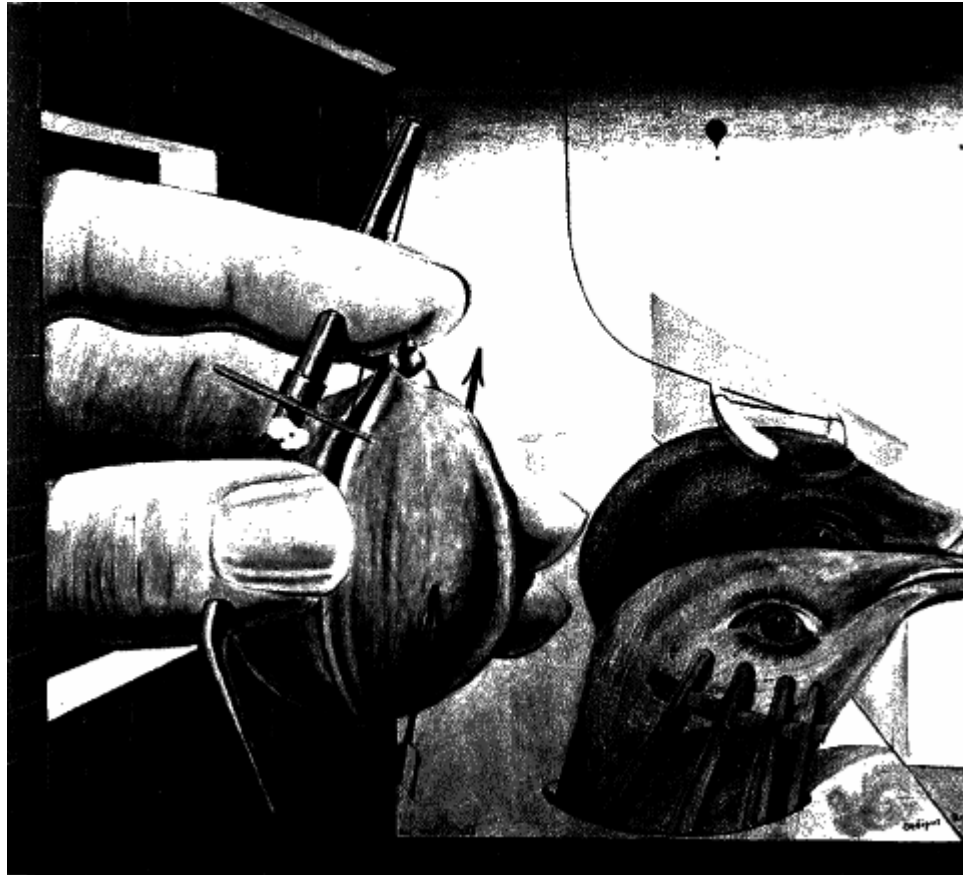
**PŘÍLOHA P XII: VYBRANÝ OBRÁZEK PRO KATEGORII:  
OBRÁZKY V 8 BITOVÉ HLOUBCE**



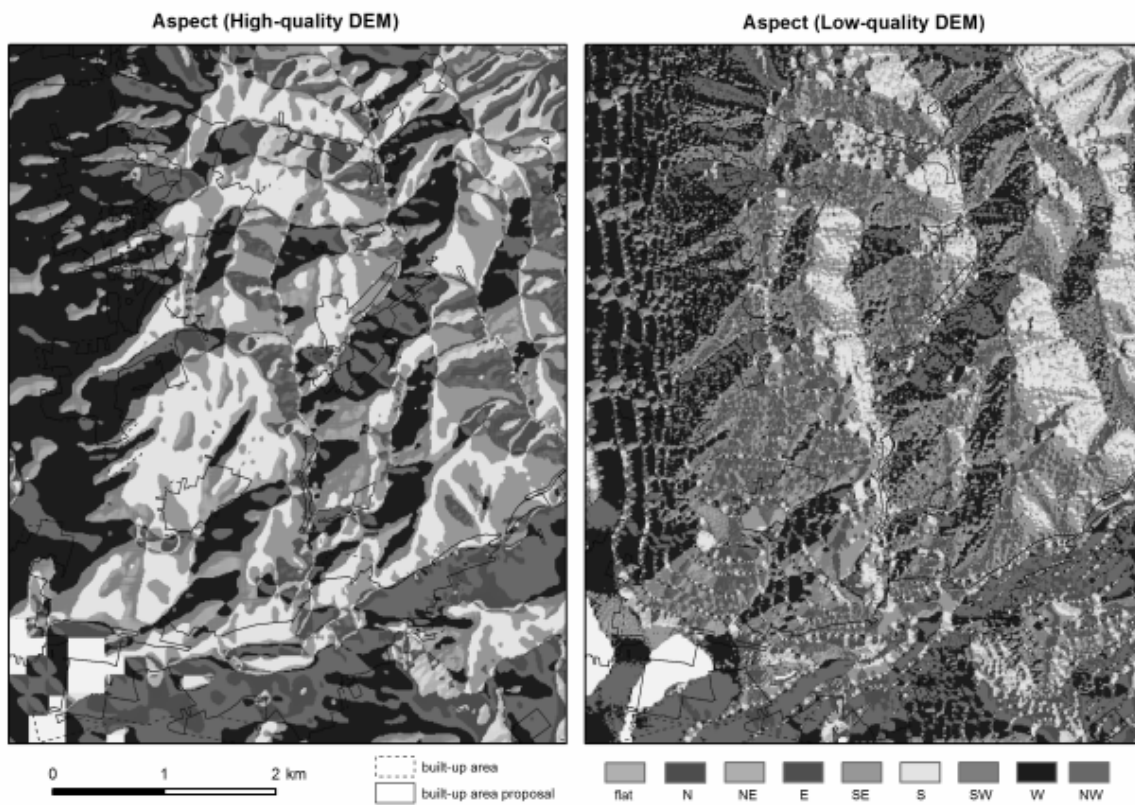
**PŘÍLOHA P XIII: VYBRANÝ OBRÁZEK PRO KATEGORII:  
OBRÁZKY VE 4 BITOVÉ HLOUBCE**



**PŘÍLOHA P XIV: VYBRANÝ OBRÁZEK PRO KATEGORII:  
OBRÁZKY V 1 BITOVÉ HLOUBCE**



**PŘÍLOHA P XV: VYBRANÝ OBRÁZEK PRO KATEGORII:  
OBRÁZKY V ODSTÍNECH ŠEDI**



## PŘÍLOHA P XVI: VYBRANÝ OBRÁZEK PRO KATEGORII: TEXT V 1 BITOVÉ HLOUBCE

— 8 —

Athènes, où l'industrie du barbier était fort en honneur, les plaideurs de la trempe de Figaro étaient rares, et si Solon avait négligé ce détail, la pratique avait bien été forcée de s'en occuper.

D'ailleurs il y avait des cas où l'observation de la loi eût été impossible en fait ou en droit. S'il s'agissait d'un enfant au berceau, d'un absent ou d'une femme, il était nécessaire que quelqu'un prit en main la cause de ce plaideur, qui ne pouvait ou ne devait pas présenter lui-même sa défense. Si l'État était directement intéressé à une affaire, si par exemple il s'agissait d'un procès de haute trahison, il ne fallait pas qu'en un cas aussi grave la défense de l'intérêt public fût abandonnée à l'initiative privée. La république désignait donc des orateurs, chargés de la représenter, et de soutenir l'accusation. Dans tous ces cas, il avait bien fallu déroger à la règle, et admettre que le plaideur se pouvait faire représenter par autrui.

De même, s'il s'agissait d'un laboureur, d'un vigneron, d'un matelot ou d'un soldat, dont la langue indocile se montrait rebelle à la parole, il devenait bien difficile d'appliquer la loi. C'eût été vraiment perdre le temps des juges, et aussi se moquer d'eux que de leur produire un plaideur absolument incapable d'exposer le premier mot de son affaire. N'était-ce pas d'ailleurs une injustice criante que de pauvres vieux soldats blanchis sous le harnais, qui dans maints combats s'étaient couverts d'une glorieuse sueur, fussent exposés sans défense aux attaques d'un vil sycophante, et se vissent ravir par une condamnation l'argent qui devait servir à leur assurer des funérailles décentes? Il avait paru juste que ce plaideur inexpérimenté pût appeler à son aide quelque ami bienveillant et disert, dont la langue officieuse voulût bien porter secours à sa détresse. Et comme les amis habiles et désintéressés étaient aussi rares à Athènes qu'ailleurs, il avait bien fallu s'adresser à quelque rhéteur ou sophiste de profession, dont on reconnaissait en secret les services par le paiement d'un honoraire.

## PŘÍLOHA P XVII: VYBRANÝ OBRÁZEK PRO KATEGORII: TEXT V ODSTÍNECH ŠEDI

spreadsheetovými aplikacemi. Výsledkem toho je, že tyto formáty můžete najít v různých platformách (Lotus DIF – Data Interchange Format a Microsoft SYLK – Symbolic LinK Format).

Většina vektorových formátů je však určena pro ukládání čárových kresb vytvořených v CAD aplikacích. CAD software se používá pro mechanické, elektronické a architektonické náčrty, elektronické výkresy a schémata, mapy a diagramy a umělecké návrhy. Složitost informací potřebných k podpoře hlavních CAD aplikací je mnohem větší než u těch, které spolupracují se spreadsheety a většinou vyžaduje mnohem složitější vektorový formát.

GCM (Computer Graphics Metafile) je příkladem všeobecného formátu založeného na výměně dat. Je to formát, který je popsán ve veřejně publikovatelném standardu. Všechny prvky v souborech s GCM formátem jsou složeny z jednoduchých objektů, jako jsou čáry, mnohoúhelníky a primitiva, a jsou k dispozici každé zobrazovací aplikaci. Velmi složité objekty se rozkládají na objekty méně složité.

Formát Autodesk AutoCAD DXF (Data eXchange Format) byl rovněž navržen s vektorovou výměnou dat, ale je kontrolován výrobcem. Byl založen jako formát podporující jednotlivé aplikace. Navíc, DXF byl speciálně navržen pro CAD informace použitelné pro podporu konstruování a projektování v mechanické, elektrické a architektonické oblasti. DXF proto podporuje nejen běžné vektorové prvky jako jsou kružnice a mnohoúhelníky, ale rovněž i složité objekty používané při CAD zobrazování, jako jsou trojrozměrné objekty, návěští a šrafování.

### Jak jsou organizovány vektorové soubory

Ačkoli se vektorové soubory ve svém složení velmi liší, většina z nich obsahuje stejnou základní strukturu: hlavička, datová sekce a značka konce souboru. Dále je potřeba uchovávat některé informace o souboru a také o tom, jak mají být data správně interpretována při zobrazování. Některé formáty umísťují tyto informace do hlavičky, některé do paty.

Vektorové soubory jsou strukturálně jednodušší než většina bitmapových souborů a jsou organizovány do datových toků. Většina informací o obsahu souboru je uložena ve vlastních datech obrazové předlohy.

Základní komponenty jednoduchého vektorového souboru jsou následující:

- Hlavička
- Data obrazové předlohy

Pokud soubor neobsahuje žádná data, bude zde přítomna pouze hlavička. Pokud existují některé informace, které se nevjdou do hlavičky, můžete je nalézt v patě souboru. V některých případech je přítomna také paleta.

- Hlavička
- Paleta
- Data obrazové předlohy

## PŘÍLOHA P XVIII: SUMARIZACE VÝSLEDKŮ TESTŮ

<b>Testovaná kategorie</b>	<b>Nejlepší kompresní algoritmus</b>	<b>Kapitola</b>
Fotografie formátu JPEG	JP2 - LW	6.1.1
Fotografie formátu RAW	JP2 - LW	6.1.2
Fotografie v odstínech šedi	JLS - LOCO-I	6.1.3
Fotografie s vysokou dynamikou barev	EXR - ZIP	6.1.4
Fotografie s vysokou dynamikou barev v odstínech šedi	EXR - ZIP	6.1.5
Obrázky ve 24 bitové hloubce	JP2 - LW	6.2.1
Obrázky v 8 bitové hloubce	PNG - D9	6.2.2
Obrázky v 4 bitové hloubce	PNG - D9	6.2.3
Obrázky v 1 bitové hloubce	JBIG - JBIG	6.2.4
Obrázky v odstínech šedi	PNG - D9	6.2.5
Text v 1 bitové hloubce	JBIG - JBIG	6.3.1
Text v odstínech šedi	JLS - LOCO-I	6.3.2