Kapitola 1

Algoritmy pro kompresi rastrových dat

Rastrová data jsou v oblasti geoinformatiky používána velmi často. Hardwarová náročnost transakčních operací prováděných s rastrovými daty však bývá značně vysoká, zpravidla vyšší než u vektorových dat srovnatelných parametrů. Tento fakt se negativně projevuje zejména u barevných rastrů s vysokým prostorovým rozlišením a barevnou hloubkou, které vznikají nejčastěji jako produkt leteckého či družicovém snímkování zemského povrchu při dálkovém průzkumu Země. Dochází tak ke zvyšování nákladů spojených s uchováváním a přenosem či editací rastrových dat. Pomocí komprimačních algoritmů můžeme tyto náklady výrazně snížit a zefektivnit tak manipulaci s rastrovými daty.

1.1 Barevné modely a vztahy mezi nimi

Barva je významným výrazovým prostředkem používaným v mapách. Z fyzikálního hlediska se jedná o vlastnost viditelného spektra elektromagnetického záření (interval 350-750 nm) vnímaného okem. Lidské oko je orgán poměrně nedokonalý, je schopno rozlišit méně než 20000 barevných odstínů a 300 odstínů šedi. Rozkladem světla pomocí hranolu vzniká spektrum, ve kterém lze identifikovat 6 základních, tzv. spektrálních barev: červenou (Red), modrou (Blue), zelenou (Green), žlutou (Yellow), azurovou (Cyan), purpurovou (Magenta). Smícháním několika odstínů základních barev lze vytvořit libovolný barevný odstín. Pomocí barevných modelů můžeme barevné odstíny klasifikovat, popsat jejich vzájemné vztahy a pravidla pro jejich míchání. Dle (Lam 1994) existují čtyři základní druhy barevných modelů:

- *Modely založené na fyziologii oka*
  Do této skupiny patří dva nejčastěji používané barevné modely RGB a CMY(K) využívající míchání trojice spektrálních barev ve zvoleném poměru.

- *Modely kolorimetrické*
  Modely vycházejí z faktu, že citlivost lidského oka na jednotlivé barevné odstíny je různá. Na základě empirického zkoumání byl vytvořen kolorimetrický model CIE. Výsledný barevný odstín vzniká jako lineární kombinace složek R, G, B (každá ze složek vynásobena specifickým koeficientem).

- *Modely komplementární*
  Modely vycházejí z teorie míchání základních a komplementárních (tj. doplňkových) barev. V praxi nejsou příliš používány.

- *Modely psychofyzikální*
  Pro míchání nepoužívají základní barvy, ale tři následující parametry: barevný tón H (Hue), sytost S (Saturation) a jas V (Value). Zástupcem jsou modely HSV a HSL.

V literatuře bývá uváděna i pátá skupina představovaná *fyzikálními modely*, jejichž zástupcem je např. model YUV používaný při přenosu televizního signálu. Tento model má spíše fyzikální než estetické či praktické opodstatnění. Při použití některých kompresních algoritmů (JPEG komprese) dochází k převodům dat mezi jednotlivými barevnými modely, cílem je dosažení vyššího kompresního poměru a menší vizuální degradace obrazu (ztrátové algoritmy). Uvedeme pouze velmi stručný popis trojice barevných modelů, na které se budeme v dalším textu odkazovat.

RGB model. V případě RGB modelu vytváříme výslednou barevnou skládáním červené, zelené a modré barvy ve vhodných poměrech (tj. Red, Green, Blue), hovoříme o tzv. *aditivním* míšení barev.

Obrázek 1.1: Znázornění modelu RGB za použití diagramu a jednotkové krychle.

Barevný odstín můžeme vyjádřit za pomocí vektoru tvořeného třemi složkami, každá nabývá hodnot v intervalu $\langle 0, 1\rangle$, pro účely výpočetní techniky ho často převádíme na diskrétní interval $\langle 0, 255\rangle$. Hodnota 0 znamená, že složka není vůbec zastoupena, hodnota 255 pak, že složka má plnou intenzitu. V tomto režimu lze vytvořit 256^3 (tj. 16777216) barevných odstínů. Smícháním všech tří složek v plné sytosti vzniká bílá barva. Model lze znázornit i pomocí jednotkové krychle. Počátek o souřadnicích [0, 0, 0] představuje černou barvu. Barevný model RGB můžeme převést na stupně šedi (Gray) za použití následujícího vztahu.

$$Gray = 0.299 \cdot R + 0.5870 \cdot G + 0.114 \cdot B \tag{1.1}$$

Čím je větší sytost barevných složek, které sčítáme, tím je výsledná barva světlejší. Jelikož přírůstek „světlosti“ není konstantní, pro namíchání správného barevného odstínu musíme často provést dodatečné zesvětlení či ztmavení obrazu za použití gama korekce. Tento barevný model je využíván u zobrazovacích zařízení (monitory, LCD displeje), není však vhodný pro tisk.

CMYK model. CMYK model je pro práci s barevnými složkami vhodnější, odpovídá praktické zkušenosti lidí při práci s barvami. Čím je větší sytost barevných složek, které sčítáme, tím je výsledná barva tmavší. Barevné složky jsou: azurová, purpurová a žlutá (tj. Cyan, Magenta, Yellow). V tomto případě hovoříme o tzv. *subtraktivním* míchání barev. Převod mezi oběma modely lze realizovat pomocí následujícího vztahu:

$$\begin{pmatrix} C \\ M \\ Y \end{pmatrix} = \begin{pmatrix} 255 \\ 255 \\ 255 \end{pmatrix} - \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{1.2}$$

Model CMYK model je používán při tisku. Černá barva (Black) vzniká soutiskem všech složek s maximální intenzitou. Běžná metoda tisku však nezaručí úplně černou barvu, ale spíše tmavě hnědou. Proto černá barva (K) tvoří čtvrtou základní barvu, je tištěna samostatně. Spotřeba barvy je v takovém případě menší než při vytváření černé barvy soutiskem základních barev.

Obrázek 1.2: Znázornění modelu CMYK za použití diagramu a jednotkové krychle.

YC_BC_R model. Popis barevných odstínů v tomto modelu je blízký skutečnému fyziologickému vnímání barev. Lidské oko vnímá barvu pomocí čípků a tyčinek: tyčinky jsou citlivé na jas, čípky na barvu. Lidské oko je vnímavější ke změně jasu než ke změně barvy. Model je představován třemi složkami: jednou jasovou (Y) a dvěma barevnými, tzv. chrominačními (C_B a C_R). Mezi RGB a YC_BC_R modely platí následující vztah:

$$\begin{pmatrix} Y \\ C_\mathrm{B} \\ C_\mathrm{R} \end{pmatrix} = \begin{pmatrix} 0.2990 & 0.5870 & 0.1140 \\ -0.1687 & -0.3313 & 0.5000 \\ 0.5000 & -0.4187 & -0.0813 \end{pmatrix} \begin{pmatrix} R \\ G \\ B \end{pmatrix} + \begin{pmatrix} 0 \\ 128 \\ 128 \end{pmatrix}. \tag{1.3}$$

Opačný převod vypadá takto:

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1.0000 & 0.0000 & 1.4020 \\ 1.0000 & -0.3441 & -0.7141 \\ 1.0000 & 1.7720 & -0.0001 \end{pmatrix} \begin{pmatrix} Y \\ C_\mathrm{B} - 128 \\ C_\mathrm{R} - 128 \end{pmatrix}. \tag{1.4}$$

Tento model je méně známý než oba předchozí, je používán v oblasti telekomunikační techniky, zejména při přenosu televizního signálu. V počítačové grafice nalezne využití při JPEG kompresi.

Barevná hloubka a rozlišení rastru. Připomeňme stručně některé další pojmy, se kterými se setkáme při práci s rastrovými soubory. Barevná hloubka rastru udává počet bitů použitých pro reprezentaci barevného odstínu. Představuje celočíselnou hodnotu, nejčastěji se používá 8, 16, 24, 32 bitová barevná hloubka. S rostoucí barevnou hloubkou narůstá i velikost rastru a zvětšuje se paměťová náročnost operací s rastry. Přechodem z 8 bitové barevné hloubky (grayscale) na barevnou hloubku 24b (true color) vzroste velikost souboru třikrát. Níže uvedené algoritmy lze použít pro všechny tři skupiny rastrů (viz tab. 1.1), každý algoritmus je však optimalizován pro jiný typ rastrů.

Rastr lze reprezentovat vhodnou strukturou, např. maticí. Rozlišení rastru představuje počet pixelů v řádku a sloupci. Matice obecně nemusí být regulární. Velikost rastrového souboru souvisí s rozlišením, roste s kvadrátem změny rozlišení. Zdvojnásobením rozlišení rastru jeho velikost vzroste čtyřikrát. Tuto rastrovou „aritmetiku“ je vhodné zvážit před tvorbou rastrového souboru, barevnou hloubku a rozlišení je nutno volit v závislosti na pozdějším využití rastru.

1.2 Komprese dat, hodnocení, metody

Komprese představuje proces kódování dat, při kterém dochází k odstraňování redundantních (tj. nadbytečných) informací s cílem umožnit efektivnější reprezentaci původních dat. Při kompresi dochází k převodu jedné reprezentace dat na jinou, která se vyznačuje nižším objemem dat. Takto vzniklá data

Typ rastru	Popis	Barevná hloubka	Interval barev. hloubky
Binární	Černobílý rastr	1b	<0, 1>
Grayscale	Stupně šedi	8b	<0, 255>
Color	Barevný rastr	16, 24, 32 b	<0, 255> pro každou složku

Tabulka 1.1: Barevné hloubky u jednotlivých skupiny rastrů.

nazýváme komprimovaná. Kompresi lze jednoznačně popsat předpisem přiřazujícím prvku v původní datové reprezentaci prvek v nové datové reprezentaci. Opačný proces, při kterém z komprimovaných dat rekonstruujeme data původní, nazýváme *dekomprese*. Cílem komprese zpravidla bývá:

- zrychlení přenosu dat,
- snížení nároků na hardware,
- efektivnější načítání a ukládání dat.

Tyto požadavky spolu vzájemně souvisejí, tvoří spojité nádoby.

1.2.1 Hodnocení kvality komprese

Kvalitu komprese můžeme hodnotit pomocí několika faktorů, uveďme nejčastěji používané:

Kompresní poměr p. Kompresní poměr p definujeme jako poměr velikosti souboru před kompresí a velikosti souboru po kompresi v bajtech (B)

$$p = \frac{s_v}{s_k}, \tag{1.5}$$

kde s_v představuje velikost vstupního souboru v bajtech a s_k velikost souboru po kompresi v bajtech. Jedná se o bezrozměrné číslo; čím je hodnota p větší, tím vyšší je míra komprese souboru, a kompresní algoritmus je účinnější. Pro $p = 1$ nedochází ke kompresi, pro $p <1$ je velikost souboru po kompresi větší než před kompresí, hovoříme o *negativní* kompresi. K negativní kompresi může dojít při nevhodném uspořádání vstupních dat, kdy algoritmus nelze efektivně použít. Hodnoty kompresního poměru se pohybují nejčastěji v rozmezí 1.5-10.

Kompresní faktor f. Kompresní faktor f je převrácenou hodnotou kompresního poměru. Udává, kolikrát je soubor po kompresi menší než soubor před kompresí. Pro $f < 1$ dochází ke kompresi, pro $f > 1$ k negativní kompresi.

$$f = \frac{1}{p} = \frac{s_k}{s_v} \tag{1.6}$$

Kompresní zisk z. Kompresní zisk z ukazuje relativní změnu velikosti souboru v průběhu komprese. Čím je tato hodnota vyšší, tím větší je kompresní poměr p. Je –li $z < 0$, dochází k negativní kompresi.

$$z = \frac{s_v - s_k}{s_v} \tag{1.7}$$

Faktor relativní komprese r. Posuzuje účinnost použitého kompresního algoritmu vzhledem k jinému algoritmu představujícímu etalon. Definujeme ho jako logaritmus poměru velikosti souboru komprimovaného etalonem a velikosti souboru komprimovaného posuzovaným algoritmem. Jako etalon lze zvolit libovolný kompresní algoritmus.

$$r = 100 \cdot \ln\left(\frac{s_e}{s_k}\right) \tag{1.8}$$

Hodnota s_e představuje velikost souboru komprimovaného etalonem. Je –li $r > 0$, posuzovaný algoritmus dosahuje vyššího kompresního poměru než etalon. Výsledná hodnota je uváděna v procentech.

Typ komprese	p	f	z	r(RLE, JPEG)
RLE	1.43	0.70	0.30	73,06
JPEG	7.69	0.13	0.87	-73.06

Tabulka 1.2: Srovnáni průměrných parametrů komprese pro RLE a JPEG kompresi; JPEG komprese dosahuje výrazně lepších výsledků.

Střední kvadratická odchylka m_r. Tato charakteristika přesnosti vyjadřuje míru rozdílu mezi hodnotami pixelů v původním a komprimovaném rastru. Uplatní se pouze u ztrátových algoritmů. Čím je hodnota m_r větší, tím větší je rozdíl mezi barevnými hodnotami v obou rastrech, a rastry jsou „méně podobné“. Označme hodnotu barevné informace v každém pixelu před provedením komprese jako z_i, hodnotu barevné informace po provedení komprese jako z'_i. Pak platí

$$m_r = \sqrt{\frac{\sum_{i=0}^{m \cdot n} (z_i - z'_i)^2}{m \cdot n}}. \tag{1.9}$$

Jmenovatel představuje celkový počet pixelů rastru. Hodnoty prvních tří kritérií je možno uvádět i v procentech.

1.2.2 Dělení kompresních algoritmů

Kompresní algoritmy dělíme nejčastěji podle těchto kritérií:

Podle vztahu ke komprimovaným datům. Podle vztahu ke komprimovaným datům algoritmy dělíme na ztrátové a bezztrátové.

1. *Bezztrátové komprimační algoritmy (Loseless Algorithms)*

   Zkomprimovaný soubor lze zpětně obnovit tak, že je totožný s původním souborem. Nedochází k žádné ztrátě informací. Bezztrátová komprese se používá na rastrové soubory méně často, bezztrátově komprimujeme texty, zdrojové kódy programů, data. Bezztrátové kompresní algoritmy dosahují menšího kompresního poměru než ztrátové, cca 2:1. Příkladem bezztrátové komprese jsou např. RLE komprese, Huffmanovo kódování, LZ77, LZW, aritmetické kódování.

2. *Ztrátové komprimační algoritmy (Lossy Algorithms)*

   Při ztrátové kompresi dochází k redukci množství informace obsažené v původním souboru. Soubor po provedené dekompresi není stejný jako původní soubor. Významná část informace zůstane zachována, oba soubory jsou si „podobné“. Ztrátové komprimační algoritmy využívají faktu, že drobné změny v obrazu lidské oko není schopno přesně zaznamenat. Rozvoj ztrátových kompresních algoritmů nastal v 80. letech v souvislosti s nástupem počítačového zpracování obrazu. Ztrátové algoritmy nelze využít pro kompresi textů, programů, lze je použít pro komprimaci statických rastrů (JPEG), animací (MPEG), zvuku (MP3). Ztrátové komprimační algoritmy dosahují vysokého kompresního poměru, až 10:1-20:1, v některých případech až 100:1.

Podle doby komprese/dekomprese. Na základě časového hlediska doby komprese/dekomprese dělíme algoritmy do dvou skupin, a to na symetrické a asymetrické.

- *Symetrické komprimační algoritmy*

  Doba komprese i dekomprese je u nich (přibližně) stejná. Příkladem komprimačního algoritmu náležícího do této skupiny může být DCT použitá v JPEG kompresi.

- *Asymetrické komprimační algoritmy*

  Doba komprese a dekomprese se liší. Ve většině případů je doba dekomprese kratší než doba komprese. Pokud se oba údaje příliš neliší (řádově v desítkách procent), jedná se o mírně asymetrické algoritmy. Pokud se doba komprese a dekomprese velmi výrazně liší (ve stovkách či tisících procentech), hovoříme o silně asymetrických algoritmech (příkladem může být fraktální komprese).

Podle počtu použitých kompresních metod. Podle počtu použitých kompresních metod dělíme komprimační algoritmy do dvou skupin, a to na homogenní a hybridní.

- *Hybridní komprimační algoritmy*

  Vznikají kombinací několika komprimačních metod za účelem optimalizace kompresního poměru. Příkladem může být ZIP či JPEG komprese.

- *Homogenní komprimační algoritmy*

  Používají pouze jednu kompresní metodu.

Podle počtu průchodů. Podle počtu průchodů dělíme kompresní algoritmy do dvou skupin, a to na jednoprůchodové a víceprůchodové.

- *Jednoprůchodové komprimační algoritmy*

  Komprimační algoritmus zpracuje vstupní soubor v jednom průchodu, představuje sekvenční způsob práce s daty. Výhodou těchto algoritmů je zpravidla jednodušší implementace, nevýhodou nižší komprimační poměr.

- *Víceprůchodové komprimační algoritmy*

  Komprimace probíhá postupně opakovaným průchodem vstupního souboru. Cílem této operace je přepočítání některých parametrů komprese tak, aby bylo dosaženo vyššího komprimačního poměru. Počet průchodů je různý, u některých metod může dosáhnout i počtu deseti (fraktální komprese). Nevýhodou je značná složitost, výhodou vyšší komprimační poměr.

1.3 Metody komprese rastrových dat

Metody komprese rastrových dat dělíme nejčastěji do čtyř skupin:

- Jednoduché metody komprese (RLE).
- Statistické metody komprese (Aritmetické kódování, Huffmanovo kódování).
- Slovníkové metody komprese (LZW, LZ-77, LZ-78).
- Transformační metody komprese (JPEG, JPEG 2000, fraktálová komprese).

1.3.1 Jednoduché metody komprese

Metody patřící do této skupiny jsou založeny na triviálních principech, nepracují se složitějším matematickým aparátem. Základní myšlenka představuje nahrazení opakujících se posloupností znaků ve vstupním souboru jedním nebo více zástupnými znaky ve výstupním souboru. Pro kompresi rastrových dat je využívána pouze jedna z níže uvedených metod, a to Run Length Encoding (RLE). Ostatní metody nalezly uplatnění při kompresi textů, dokumentů či programů, kde dosahují vyšších komprimačních poměrů než u rastrových souborů.

Run Length Encoding (RLE)

Algoritmus je založen na nahrazování posloupnosti opakujících se znaků jedním (popř. dvěma) zástupným znakem v případě, kdy je počet opakujících se znaků větší než zadané kritérium. Aby byl algoritmus efektivní, počet opakování by měl být větší než tři, zpravidla to bývají nejméně čtyři znaky. V opačném případě by mohlo dojít k negativní kompresi. Jedná se o symetrickou metodu proudového kódování (vstupnímu proud dat je v reálném čase generován odpovídající výstupní proud dat) s jedním průchodem souboru. Postup je používán převážně ke komprimaci binárních rastrů, lze ho použít i pro barevné rastry s opakujícími se plochami stejných barevných odstínů. Podrobnosti viz [Žára04].

Princip komprese. V souboru hledáme pravidelně se opakující posloupnosti znaků. Nalezneme –li je, nahradíme jejich výskyt dvojicí znaků. První znak označený I_c představuje identifikátor komprese, druhý znak počet výskytu opakujících se znaků v posloupnosti označovaný n. Znak I_c představuje symbol, který se v souboru nevyskytuje. V případě rastrových souborů je maximální hodnota barevné hloubky rovna 255. Jako I_c je možné použít vyšší hodnotu, např. 256.

```
Vstupní data: 172 126 126 126 126 126 126  126 126 195 195 21 21 21 21
Výstupní data: 172 256 126 8 195 195 256 21 4
```

Metoda RLE má řadu modifikací. Jedna z variant umožňuje efektivnější způsob komprese, kdy do souboru nepřidáváme dva znaky, ale pouze jeden znak představující zarážku zvýšenou o počet opakování znaku n. Novou hodnotu H určíme jako

$$H = I_c + n. \tag{1.10}$$

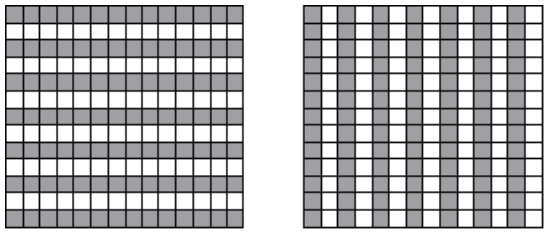
Aplikujeme –li ji na posloupnost vstupních dat, bude výsledek vypadat takto:

```
Výstupní data: 172 255+8 126 195 255+4 21  ->  172 263 126 195 259 21
```

Postup RLE komprese jednoho řádku je znázorněn v tabulce tab. 12.1.

Princip dekomprese. Dekomprese probíhá postupným procházením položek souboru. Testujeme, zda je hodnota prvku větší než I_c. Pokud ne, přidáme znak do dekomprimovaného souboru a načítáme další znak. Pokud ano, nastavíme příznak provádění komprese a přidáme do dekomprimovaného souboru n opakujících se hodnot a zrušíme příznak dekomprese. Postup dekomprese jednoho řádku je znázorněn v tabulce tab. 12.2.

Účinnost komprese. Run Length Encoding dosahuje nejlepšího komprimačního poměru u binárních rastrů, při jejichž kompresi se také nejčastěji používá. Účinnost komprese ovlivňuje také velikost souboru, lepších výsledků dosahuje v případě dlouhých souborů (možnost existence delších sekvencí opakujících se znaků). Důležitou roli hraje i orientace vzoru, který budeme komprimovat. Kompresí vodorovných linií dosáhneme vysokého komprimačního poměru, komprese svislých linií je naopak značně neefektivní, v takto definované variantě RLE algoritmu nedojde k žádné kompresi, viz obr. 1.3.



Obrázek 1.3: RLE komprese vodorovných linií po řádcích je značně efektivní (opakují se stejné hodnoty), komprese svislých linií po řádcích je neefektivní (hodnoty se střídají).

Modifikace algoritmu RLE. Algoritmus RLE byl proto modifikován tak, aby se jeho efektivita zvýšila i při práci s vodorovnými liniemi, resp. s pravidelně opakujícími se vzory nezávisle na jejich orientaci. Výsledkem je nesymetrická dvouprůchodová varianta. Lze ji realizovat dvěma způsoby:

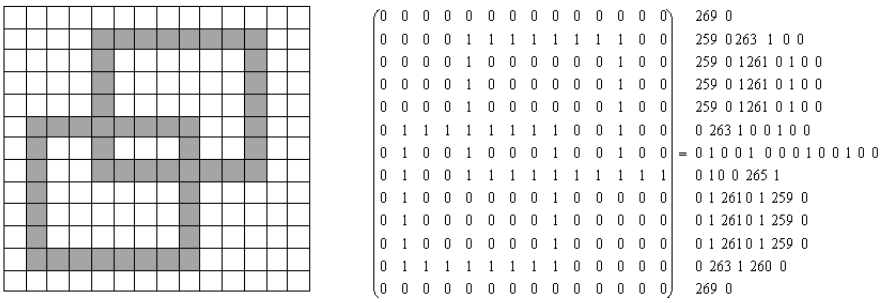
1. *Kompresí po řádcích/sloupcích*

   V prvním průchodu provedeme RLE kompresi po řádcích, v druhém kompresi po sloupcích obrazové matice. Zvolíme variantu a vyšším kompresním poměrem.

2. *Vyhledáním opakujících se vzorů*

   V prvním průchodu je provedena analýza obrázku, ve kterém jsou vyhledány pravidelně se opakující vzory. Při druhém průchodu zapisuje místo polohy opakujících se pixelů polohu opakujících se vzorů. Varianta je však poměrně náročná na implementaci.

Nevýhodou RLE komprese je nižší kompresní poměr (cca. 1.7 -2). Tato metoda je základem komprese používané ve formátech BMP, PCX, používá se i ve zvukovém formátu WAV. Algoritmus je patentován, nemůže být volně užíván. Další podrobnosti viz [Žára04].



Obrázek 1.4: Ukázka RLE komprese binárního rastrového souboru.

1.3.2 Metody statistické komprese

Tyto kompresní algoritmy patří mezi nejpoužívanější. Lze je charakterizovat jako algoritmy víceprůchodové, bezztrátové, mírně asymetrické. Jsou založeny na statistické analýze znaků vstupního souboru způsobujícím asymetričnost. Do této skupiny komprimačních algoritmů patří i Huffmanovo kódování a Shannon–Fanovo kódování.

Četnost výskytu znaků. Základním analyzovaným parametrem je četnost výskytu znaku i označovaná v dalším textu jako p_i. Tato hodnota může být určena obecně pro jakýkoliv jazyk. Existují tabulky četnosti výskytu znaků v různých jazycích. Tento údaj použil již v roce 1800 Morse při konstrukci Morseovy abecedy, kdy přiřadil znakům s větší četností výskytu kratší kódy a znakům s menší četností výskytu delší kódy. Dosáhl tak efektivnějšího přenosu dat.

Znak	Kód	Znak	Kód	Znak	Kód	Znak	Kód
A	.-	H	....	N	-.	U	..-
B	-...	CH	----	O	---	V	...-
C	-.-.	I	..	P	.--.	W	.--
D	-..	J	.---	Q	--.-	X	-..-
E	.	K	-.-.	R	.-.	Y	-.--
F	..-.	L	.-..	S	...	Z	--..
G	--.	M	--	T	-		

Tabulka 1.3: Morseova abeceda. Znaky s nejčastější frekvencí výskytu A, E, I, T mají nejkratší kódy.

Pro kompresi konkrétního souboru však tento obecný údaj nemá prakticky žádný smysl. Komprimujeme totiž obecnou posloupnost znaků formálního jazyka, v každé takové posloupnosti budou četnosti různé. Četnost výskytu znaku p_i určíme jako součet všech jeho výskytů v posloupnosti znaků.

1.3.2.1 Huffmanovo a Shannon-Fanovo kódování

Algoritmus Huffmanova kódování byl navržen Davidem Huffmanem v roce 1952. Je založen na tvorbě prefixového kódu vzniklého analýzou vstupního souboru. Prefixový kód je kódem s minimální délkou, žádný kód znaku nezačíná kódem (tj. není prefixem) znaku jiného. Kompresní algoritmus využívá myšlenky, že znaky původního souboru mohou být reprezentovány různě dlouhým kódem představovaným posloupností bitů. Tento kód nazýváme VLC (=variable length code). Snaží se nalézt optimální kód, aby byl dosažen co největší kompresní poměr. Znakům s větší pravděpodobností výskytu p_i přiřazuje kratší kódy k_i, znakům s menší pravděpodobností p_i delší kódy k_i. Je –li N celkový počet znaků vstupního souboru, délku výsledného kódu d_i můžeme dále [Čape00] určit jako

$$d_i = \sum_{i=1}^{N} p_i \cdot k_i.$$

Lze dokázat, že délka takto vzniklého kódu bude vždy menší než délka původního kódu. Existují dvě varianty Huffmanova kódování, které postupně popíšeme:

- Statické Huffmanovo kódování.
- Adaptivní Huffmanovo kódování.

Nejprve se seznámíme s Shannon–Fannovým kódováním. Princip tvorby Shannon–Fannova kódu je podobný tvorbě Huffmanova statického kódu.

Shannon–Fanovo kódování. Představuje dvouprůchodový kompresní algoritmus, výsledkem komprese je prefixový strom (viz dále). Tvorba Shannon–Fanova kódu probíhá takto: V prvním průchodu je prováděna analýza četností znaků vstupního souboru, každému znaku přiřadíme hodnotu absolutní četnosti. Tyto uspořádané dvojice seřadíme sestupně podle hodnot četnosti. Výsledkem je tabulka, každému znaku p_i je přiřazena hodnota k_i.

```
Vstupní data: 172 126 126 126 126 126 126 126 126 195 195 21 21 21 21
```

V následující tabulce (tab. 1.4) jsou zobrazeny uspořádané dvojice znak-četnost odpovídající vstupním datům.

Hodnota	p_i
126	8
21	4
195	2
172	1

Tabulka 1.4: Uspořádané dvojice znak-četností.

Druhý krok probíhá opakovaně. Tabulku rozdělíme na dva intervaly tak, aby rozdíly sum četností p_i v obou rozdělených intervalech byly minimální. Lze to zapsat symbolicky jako

$$|\sum_{i=1}^{j} p_i - \sum_{k=j+1}^{N} p_k| = min. \tag{1.11}$$

Prvkům ležícím v prvním intervalu přiřadíme přiřadíme kód 0, prvkům ležícím v druhém intervalu kód 1. Výsledek nalezneme v tab. 1.5.

Hodnota	p_i	S-F kód
126	8	8
21	4	1
195	2	1
172	1	1

Hodnota	p_i	S-F kód
126	8	0
21	4	10
195	2	11
172	1	11

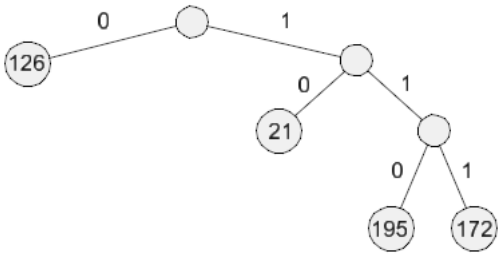
Hodnota	p_i	S-F kód
126	8	0
21	4	10
95	2	110
172	1	111

Tabulka 1.5: Postup tvorby Shannon–Fanova kódu.

Postup rekurzivně aplikujeme na intervaly, které obsahují více než jeden prvek. Kód každého prvku skládáme z posloupnosti nul a jedniček. Takto vzniklý kód nazýváme Shannon–Fanovým kódem. Všimněme si, že se skutečně jedná o *prefixový kód*, kód žádného znaku nezačíná kódem předcházejícího znaku. To zaručuje jednoznačnou dekódovatelnost komprimovaných dat. Na výstupu dostáváme místo každého z písmen posloupnost bitů (nikoliv bajtů). Znak tvořený 1 bajtem (=8 bitů) byl zakódován menším počtem bitů, znamená úsporu místa.

```
Výstupní data: 111 0 0 0 0 0 0 0 0 110 110 10 10 10
```

Shannon–Fanův kód lze vyjádřit pomocí tzv. *prefixového stromu*. Jedná se o binární strom. Znaky vstupního souboru tvoří listy, výsledný kód znaku tvoří cesta do příslušného listu z kořene stromu, viz obr. 1.5.



Obrázek 1.5: Shannon–Fanův prefixový strom.

Shannon–Fanův kód nepředstavuje optimální prefixový kód, výsledkem není optimální prefixový strom; tento strom navíc není vyvážený. Další podrobnosti v [Čape00].

Huffmanovo statické kódování. Jedná se opět o dvouprůchodový kompresní algoritmus. Na rozdíl od Shannon-Fanova kódu představuje Huffmanův kód optimální prefixový kód. Takto vzniklý kód je podobný Shannon-Fanově kódu (pro některé konfigurace dat mohou být oba kódy stejné). Huffmanovo kódování tedy generuje kód s minimální délkou.

Huffmanův kód reprezentovaný Huffmanovým stromem je vytvářen postupem zdola nahoru, tj. od listů ke kořeni. Listy tvoří prvky s nejnižší četností výskytu, mají nejdelší kód, na které napojujeme další prvky s vyšší četností, které mají kratší kód. Postup tvorby vychází ze Shannon-Fanova algoritmu. První krok je podobný, představuje seřazení znaků ve vstupním souboru podle relativní četnosti.

Hodnota	p_i
126	0,53
21	0,27
195	0,13
172	0,07

Tabulka 1.6: Přehled relativních četností znaků v původním souboru.

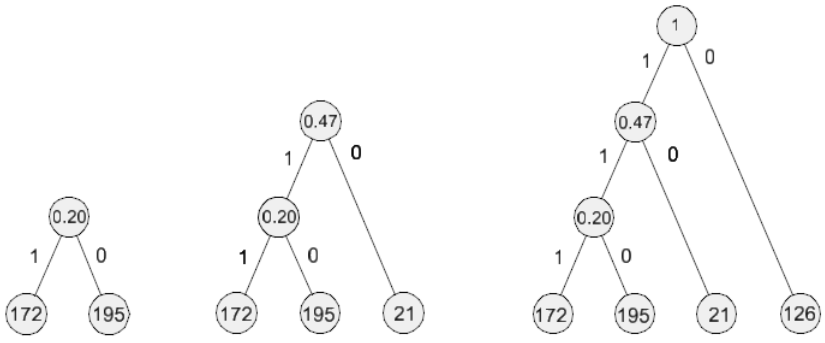
Vlastní postup budování stromu je jiný, uveďme jednotlivé kroky, viz [Čape00]:

1. Nalezneme dva uzly představované znaky s nejnižšími relativní četnostmi, budou tvořit listy nově vytvářeného Huffmanova stromu.
2. Vytvoříme předka této dvojice uzlů (kořen), ohodnocení předka představuje součet ohodnocení potomků.
3. Nalezneme dvojici uzlů s nejnižším ohodnocením. Může být tvořena kombinacemi: (1) uzel-list, (2) list-list.
4. Vytvoříme předka této dvojice (nový kořen), jeho ohodnocení je rovno součtu ohodnocení potomků.
5. Opakujeme body 3) a 4) tak dlouho, dokud součástí stromu nejsou všechny znaky, (tj. nedojde ke spojení dílčích stromů).

Ohodnocení kořene je rovno součtu relativních četností všech uzlů, je vždy rovno 1. Takto vytvořený Huffmanův strom je přidáván do výstupního souboru. Soubor dále obsahuje posloupností bitů (tj. Huffmanův kód) odpovídající jednotlivým položkám vstupních dat.

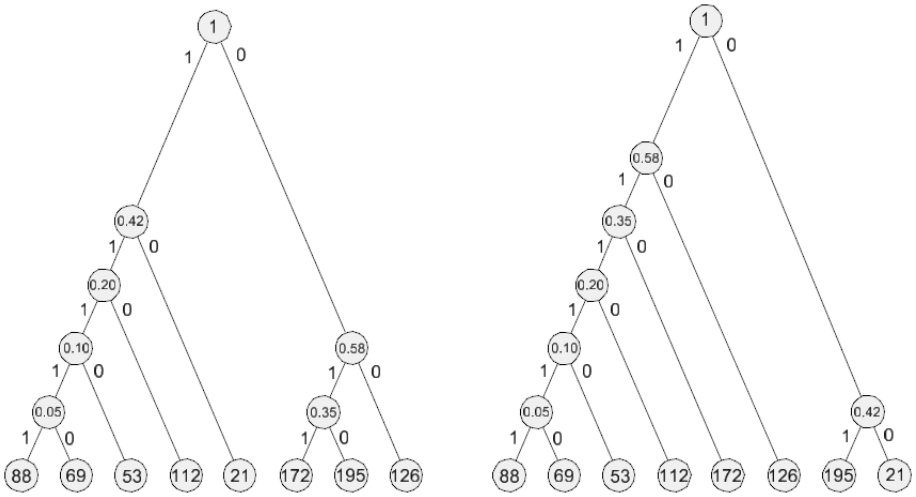
```
Výstupní data: 111 0 0 0 0 0 0 0 0 110 110 10 10 10
```

V tomto případě jsou Huffmanův i Shannon–Fanův strom identické.



Obrázek 1.6: Postup tvorby Huffmanova prefixového stromu.

Pokud se při komprimaci vyskytnou uzly se stejnou relativní četností (popř. je součet ohodnocení kořene a přidávaného uzlu roven součtu ohodnocení přidávaného a následujícího uzlu), existuje několik variant tvorby Huffmanova stromu. Výsledný strom je však v každém případě optimální. Podívejme se na následující příklad. Odpovídající prefixové stromy jsou znázorněny na obr. 1.7.



Obrázek 1.7: Různé varianty Huffmanova prefixového stromu pro stejná data.

Hodnota	p_{ri}
126	0,23
21	0,22
195	0,20
172	0,15
112	0,10
53	0,05
69	0,03
88	0,02

Tabulka 1.7: Přehled relativních četností znaků v původním souboru.

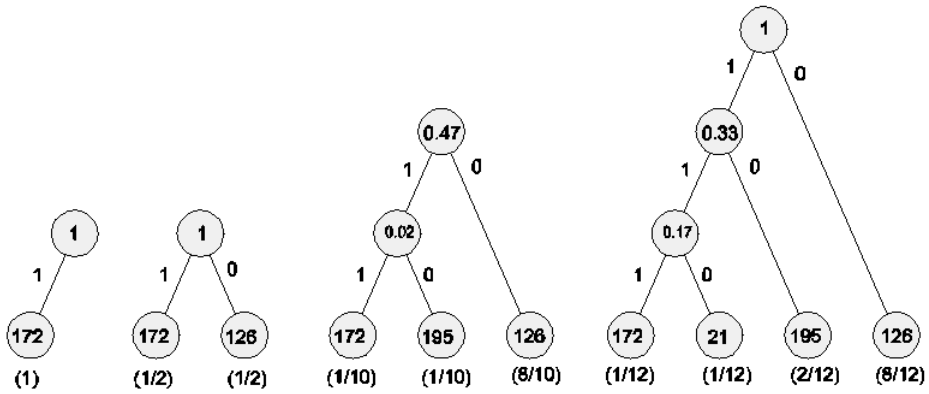
Dekomprese probíhá načítáním komprimovaného souboru po bitech. Procházíme při ní vytvořený prefixový strom, a to vždy shora dolů, tj. od kořene k listům. Jelikož se jedná o prefixový kód, je zaručena jeho jednoznačná dekódovatelnost.

Načteme první hodnotu, je –li rovna nule, přejdeme do pravého podstromu. Je –li hodnota rovna 1, přejdeme do levého podstromu. Načteme další hodnotu a postupujeme stejným způsobem, dokud nedorazíme do některého z listů. V něm kód nahradíme příslušným znakem. Poté provádíme prohledávání prefixového stromu stejným způsobem opět od jeho kořene.

Adaptivní Huffmanovo kódování. Adaptivní Huffmanovo kódování je na rozdíl od statického Huffmanova kódování jednoprůchodové. Výhodou je nižší složitost algoritmu realizujícího kompresi. Neprovádí samostatnou statistickou analýzu vstupního souboru, v prvním kroku nastaví všem znakům četnost 0. V průběhu komprese se však údaje o četnosti zpracováním každého znaku přepočítávají a přegenerovává se i vytvořený strom. Poskytuje stejný kód jako statické kódování, vytváří ho však ve „více krocích“. V tab. 1.8 nalezneme přehled relativních četností aktualizovaných při každém načtení nového znaku původní posloupnosti.

Změna relativních četností způsobuje přegenerovávání aktuálního prefixového stromu po každém načteném znaku. Tato operace je při rozsáhlejších datech poměrně náročná. Adaptivní Huffmanovo kódování je proto „více“ asymetrické než statické kódování, má větší paměťovou i implementační náročnost. Výhodou je možnost změn statistických parametrů za chodu a ovlivňování výsledků komprese.

Existuje i upravená varianta Huffmanova kódování, která neprovádí aktualizaci stromu po každém



Obrázek 1.8: Fáze tvorby prefixového stromu pro n_{r1}, n_{r12}, n_{r10} a n_{r12}

načtení znaku, ale pouze v okamžiku, kdy je nutné. Jedná se o takový znak, při jehož přidáním vzroste relativní četnost znaků v levém podstromu tak, že je větší než v pravém.

Prvek	n_{r1}	n_{r2}	n_{r3}	n_{r4}	n_{r5}	n_{r6}	n_{r7}	n_{r8}	n_{r9}	n_{r10}	n_{r11}	n_{r12}	n_{r13}	n_{r14}	n_{r15}
172	1	1/2	1/3	1/4	1/5	1/6	1/7	1/8	1/9	1/10	1/11	1/12	1/13	1/14	1/15
126	0	1/2	2/3	3/4	4/5	5/6	6/7	7/8	8/9	8/10	8/11	8/12	8/13	8/14	8/15
195	0	0	0	0	0	0	0	0	0	1/10	2/11	2/12	2/13	2/14	2/15
21	0	0	0	0	0	0	0	0	0	0	0	1/12	2/13	3/14	4/15

Tabulka 1.8: Relativní četnosti jednotlivých prvků při jejich postupném načítání.

Aktualizaci stromu provedeme tak, že zaměníme tento vrchol s vrcholem se stejnou četností takovým, který se nachází na stromu co nejvíce vpravo a leží na co nejnižší úrovni. Následně postupujeme stromem směrem ke kořeni a provádíme inkrementaci příslušných uzlů o 1. Dojde –li k tomu, že levý podstrom má větší relativní četnost než podstrom pravý, aktualizujeme strom stejným způsobem. Algoritmus má menší časovou složitost než statické Huffmanovo kódování. Postup nazýváme FGK (Faller - Gallagher - Knuth) algoritmus.

Účinnost komprese a použití. Huffmanovo kódování je používáno pro kompresi ve formátech JPEG a MPEG či ZIP (Huffmanovo kódování + LZW), ARJ jako doplňková metoda. Důležitou roli ovlivňující účinnost komprese hraje znalost rozložení četnosti znaků. Nepřesné určení těchto hodnot má negativní vliv na kompresní faktor, nejkratší kódy nemusí být přidělovány nejčastěji používaným znakům. Huffmanovo kódování dosahuje nejlepšího kompresního poměru, pokud jsou četnosti znaků násobky 2. Dlouho bylo považováno za nejlepší algoritmus pro kompresi textových dat, v poslední době je vytlačováno metodami slovníkové komprese. Huffmanovo kódování není patentově chráněno, proto je často využíváno.

1.3.3 Metody slovníkové komprese

Slovníkové metody jsou založeny na vyhledávání opakujících se posloupností znaků v komprimovaných datech. Takovouto posloupnost se snaží v každém dalším budoucím výskytu nahradit nějakým kratším kódem. Na rozdíl od předchozí skupiny algoritmů nemusíme mít k dispozici žádné statistické informace o komprimovaném souboru. Metody slovníkové komprese dělíme podle metodiky komprese do dvou skupin:

1. *Metoda posuvného okna*

Analyzujeme, zda se posloupnost znaků určité délky již nevyskytovala ve vstupních datech. Pokud ano, je místo ní umístěn odkaz na první (popř. předchozí) místo výskytu. Do této skupiny patří algoritmy LZ-77, LZB, či LZH.

2. *Metoda rostoucího slovníku*

   Ze vstupních dat je dynamicky vytvářen slovník. Analyzujeme, zda se posloupnost znaků nenachází ve slovníku. Pokud ano, je na jeho místo umístěn kód, pod kterým se ve slovníku nachází. Do této skupiny algoritmů patří LZ-78 a LZW.

Podle způsobu tvorby slovníku dělíme metody slovníkové komprese do dvou skupin:

1. *Metoda statické slovníkové komprese*

   Slovník je vytvořen před počátkem komprese, v průběhu komprese se nemění. Je součástí komprimovaného souboru.

2. *Metoda adaptivní slovníkové komprese*

   Slovník je vytvářen v průběhu komprese souboru na základě vstupních dat. Není součástí komprimovaných dat, algoritmus ho umí zpětně z komprimovaných dat vytvořit. Do této skupiny algoritmů patří LZW.

1.3.3.1 Algoritmus LZ-77

Algoritmus byl navržen v roce 1977, název představuje akronym příjmení autorů: Abrahama Lempela a Jakoba Ziva. Metoda je založena na nahrazování opakujících se posloupností znaků (tzv. *předpon*) kódy odkazujícími na jejich výskyt v předchozím textu. Nalezneme –li takovou posloupnost znaků, nahradíme jejich výskyt trojicí znaků. První znak představuje vzdálenost počátku předpony od aktuálně prohledávané pozice, druhý znak délku předpony a třetí znak první následující znak za předponou (tj. první „neshodný znak“).

Princip algoritmu LZ-77 lze snadno ilustrovat na následujícím příkladu, kdy se budeme snažit zakódovat slovo: "KOKOS". Výsledkem bude posloupnost znaků "KO22S", kterou lze interpretovat takto. Předpona "KO" začíná o 2 znaky vlevo vzhledem k aktuální pozici kurzoru, její délka činí dva znaky, první neshodný znak je "S". Níže uvedený popis algoritmu LZ-77 vychází ze [Stan04].

Postup komprese. Metoda LZ-77 využívá posuvného okna (Sliding Window), které se pohybuje nad vstupním souborem ve směru zleva do prava. Jeho délka bývá různá, závisí na způsobu implementace, nejčastěji se používá hodnota 2^{15} bajtů, tj. 32 768 znaků. Délku okna označme d, počet znaků v souboru označme n.

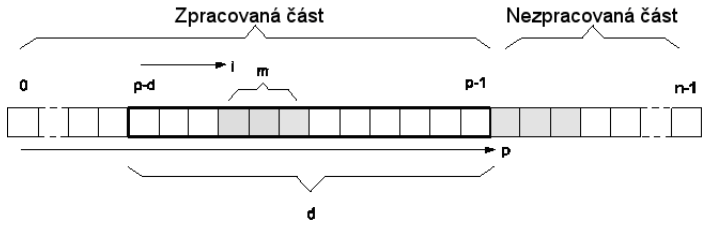
Při metodě LZ-77 není z časových důvodů prohledávána celá již zpracovaná část vstupního souboru, ale pouze posloupnost znaků o délce d (tj. odpovídající délce okna) od aktuální pozice. Hledáme řetězec s maximální délkou m (tato hodnota představuje počet shodných znaků), který je předponou dosud nezpracované části souboru. Aktuální pozici v textu označme p, tato pozice také označuje počátek dosud nezpracované části souboru. Představuje současně znak, který je počátkem (možné) maximální předpony, jež se současně vyskytuje i v okně, viz obr. 1.9.

Pozici počátku shodného řetězce vzhledem k počátku okna označíme i. Aktuální poloha okna je dána souřadnicemi $\{p-d, p-1\}$, dosud nezpracovaná část souboru souřadnicemi $\{p, n-1\}$. Relativní pozici dp počátku shodného řetězce vzhledem k p můžeme vyjádřit jako

$$dp = d - i + 1. \tag{1.12}$$

Hodnota z představuje poslední zpracovaný znak. V případě nalezení shody ($m > 0$) ho lze interpretovat jako první „neshodný“ znak, viz. výše. V následujícím kroku bude z představovat poslední znak sliding window (viz. tab. 2.9). Platí, že

$$z = p + m. \tag{1.13}$$



Obrázek 1.9: Poloha slide window na pozici p v prohledávaném souboru.

Pokud nebyla nalezena žádná shoda mezi nezpracovanou částí textu a sliding window, $dp = 0$, $z = p$, $m = 0$, sliding window posuneme o jeden znak vpravo. Pokud byla nalezena shoda v m znacích, posuneme sliding window o $m + 1$ znaků. Výstupem LZ-77 je uspořádaná posloupnost tří údajů: hodnota *dp,* hodnota m a znak z, které jsou zapsány na výstup (viz tab. 1.9). Novou aktuální pozici p v řetězci tedy určíme jako

$$p = z + 1 = p + m + 1. \tag{1.14}$$

Stejným způsobem zapíšeme na výstup polohy a délky všech pozic u případných dalších nalezených shod. Podívejme se na postup komprese následujících dat pro $d = 6$.

```
Vstupní data: Leteljelennadjetelem
Výstupní data l e t 2 1 l j 3 2 e n 1 1 a d j 6 1 t 2 1 l 4 1 m
```

Text																			dp	m	z
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	0	0	l
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	0	0	e
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	0	0	t
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	2	1	l
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	0	0	j
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	3	2	e
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	0	0	n
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	1	1	a
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	0	0	d
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	0	0	j
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	6	1	t
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	2	1	l
1	e	t	e	l	j	e	l	e	n	n	a	d	j	e	t	e	l	e	m	4	1	m

Tabulka 1.9: Komprese algoritmem LZ-77.

Postup dekomprese. Postup dekomprese je podobný jako v případě komprese. Používá stejné okno pohybující se nad posloupností znaků. Jakmile okno narazí na posloupnost trojice znaků identifikujících polohu a velikost předpony, zkopíruje místo nich příslušnou předponu. Postup je opakován, dokud neprovedeme nahrazení všech předpon.

Účinnost komprese a použití. LZ-77 v kombinaci s Huffmanovým kódováním označujeme jako metodu *deflate*. Je základem komprimačních programů ZIP a PKZIP a představuje jeden z nejčastěji používaným komprimačních algoritmů vůbec. LZ-77 je používán i v oblasti počítačové grafiky, dosahuje poměrně dobrých kompresních poměrů u dat obsahujících pravidelně se opakující posloupnosti znaků (tj. pravidelné vzory). Je součástí formátu PNG, který byl open-source komunitou vytvořen jako

náhrada patentově chráněného formátu GIF (patentová práva na GIF vypršela 08.2006) převážně pro grafická data v prostředí Internetu.

Značnou výhodou oproti jiným formátům (např. JPEG) je fakt, že i přes velký kompresní poměr je LZ-77 bezztrátový, a nedochází tak k vizuální degradaci rastrových dat. Algoritmus je silně asymetrický, má poměrně značnou časovou složitost (ovlivněno opakovaným hledáním předpony). Pro efektivnější hledání předpony v okně je možno použít Morris-Prattův algoritmus. Účinnost komprese ovlivňuje délka okna, u dlouhých oken dosáhneme vyššího kompresního poměru, značně však vzroste doba komprese. Menší délka okna dobu komprese výrazně zkrátí, dosažený kompresní poměr však bude nižší. Implementace je poměrně složitá, vzhledem k rozsahu publikace ji nebudeme uvádět.

1.3.3.2 Algoritmus LZW

Algoritmus představuje modifikaci LZ-77. Jeho jméno je opět tvořeno akronymem tvůrců: Abrahama Lempela, Jakoba Ziva a Terryho Welche. Vzhledem k době objevení je někdy také nazýván LZ-78. Algoritmus používá dynamicky vytvářený slovník, jehož velikost se v průběhu komprese mění, a obsah se adaptivně přizpůsobuje komprimovaným datům. Slovník nemusí být ke komprimovaným datům přidáván, vytváří se automaticky při dekompresi. Součástí výstupního souboru však musí být seznam všech znaků, které se vyskytují v souboru.

Postup komprese. Komprese je tvořena několika kroky. Ze vstupního souboru jsou načítány znaky, z nich je složen takový řetězec, jehož předponu tvoří fráze již uložená ve slovníku. Řetězec je o jeden znak delší než fráze, tvoří ho fráze + nově načtený znak. Tímto řetězcem nahradíme původní frázi ve slovníku, řetězec se stává novou frází. Fráze jsou označovány kódy, výskyt frází ve vstupním souboru nahrazujeme těmito kódy, což je základním principem slovníkové komprese. S rostoucí délkou frází tak nahrazujeme kódem stále delší řetězce a algoritmus dosahuje vyššího kompresního poměru.

Naplnění slovníku:

z	r	r+z	Fráze ve slovníku? (f==r+z?)	Kód c	Přidat do slovníku? (r+z → f)	Přidání f	r=r+z	Výstup $v=c_r$
-	-	-	-	0	-	1	-	-
-	-	-	-	1	-	7	-	-
-	-	-	-	2	-	2	-	-
-	-	-	-	3	-	6	-	-
-	-	-	-	4	-	5	-	-
-	-	-	-	5	-	9	-	-

Komprese souboru:

Načítáme z	r	r+z	Fráze ve slovníku? (f==r+z?)	Kód c	Přidat do slovníku? (r+z → f)	Přidání f	r=r+z	Výstup $v=c_r$
1	-	-	-	-	Ne	-		
7	1	17	Ne	6	Ano	17	-	**0**
2	7	72	Ne	7	Ano	72	-	**1**
1	2	21	Ne	8	Ano	21	-	**2**
2	1	12	Ne	9	Ano	12	-	**0**
6	2	26	Ne	10	Ano	26	-	**2**
1	6	61	Ne	11	Ano	61	-	**3**
2	1	12	Ano	9	Ne	-	12	-
6	12	126	Ne	12	Ano	126	-	**9**
1	6	61	Ano	11	Ne	-	61	-
2	61	612	Ne	13	Ano	612	-	**11**
6	2	26	Ano	10	Ne	-	26	-
1	26	261	Ne	14	Ano	261	-	**10**
2	1	12	Ano	9	Ne	-	12	-
6	12	126	Ano	12	Ne	-	126	-
1	126	1261	Ne	15	Ano	1261	-	**12**
2	1	12	Ano	9	Ne	-	12	-
6	12	126	Ano	12	Ne	-	126	-
1	126	1261	Ano	15	Ne	-	1261	-
2	1261	12612	Ne	16	Ano	12612	-	**15**
6	2	26	Ano	10	Ne	-	26	-
...	...	...	...	...		...	...	**...**

Tabulka 1.10: Komprese dat algoritmem LZW.

Načítaný znak označme z, posloupnost znaků tvořících řetězec r, frázi uloženou ve slovníku f, kód fráze ve slovníku c (udává pozici fráze ve slovníku). Postup komprese vycházející ze [Stan04] lze vyjádřit takto:

1. Do slovníku přidáme všechny znaky z vyskytující se ve vstupním souboru, hodnotě r přiřadíme prázdný řetězec.
2. Dokud nepřečteme všechny znaky vstupního souboru, opakujeme:
   (a) Načteme aktuální znak z ze vstupního souboru.
   (b) Testujeme, zda se ve slovníku vyskytuje fráze f taková, že
   $$f = z + r. \tag{1.15}$$
   (c) Pokud se taková fráze f ve slovníku vyskytuje, přidáme do řetězce r aktuální znak z. Bude platit, že
   $$r = r + z. \tag{1.16}$$
   Dále postupujeme bodem a).
   (d) Pokud se taková fráze f ve slovníku nevyskytuje, přidáme $z + r$ do slovníku, do výstupního souboru zapíšeme kód c znaku r, nastavíme $r = z$ a pokračujeme bodem a).

Podívejme se na postup LZW komprese pro následující vstupní data. Jednotlivé kroky jsou uvedeny v tab. 1.10. Připomeňme, že sloupec f představuje frázi přidávanou do slovníku, sloupec v kódy zapsané do výstupního souboru.

```
Vstupní data: 172 126 126 126 126 126 126 126 126 195 195 21 21 21 21
```

Tato informace by nebyla postačující pro dekompresi souboru, data musí být doplněna seznamem všech znaků vyskytujících se ve vstupním souboru (představují ho fráze přidávané do slovníku ve fázi jeho naplnění). Oba sloupce představující výstupní data jsou v tabulce 2.9 zvýrazněny.

```
Výstupní data: {1,7,2,6,5,9} a 0 1 2 0 2 3 9 11 10 12 15...
```

Postup dekomprese. Postup dekomprese je analogický, vychází z některých vlastností komprese. Zopakujme, že při kompresi jsou z načítaných znaků skládány řetězce tvořící fráze; fráze představuje řetězec, ke kterému přidáme počáteční znak následujícího řetězce. Do výstupního souboru je zapsán kód řetězce, do slovníku fráze s novým kódem. První znak aktuální fráze a poslední znak předchozí fráze jsou totožné. Tyto vlastnosti jsou viditelné v tab. 1.10.

V prvním kroku dekomprese opět naplníme slovník všemi znaky vyskytujícími se v komprimovaném souboru. Postupně dekódujeme jednotlivé fráze, dokud není dosaženo konce souboru. Je –li fráze s aktuálním kódem již ve slovníku, lze ji přímo dekódovat. Zapíšeme frázi do výstupního souboru a přidáme do slovníku řetězec tvořený dekódovanou předchozí frází + prvním znakem dekódované aktuální fráze (srovnej s kompresí).

Situace, kdy fráze s aktuálním kódem dosud není ve slovníku, nastane pouze v případě, pokud při kompresi na výstup umístíme slovo odpovídající poslední přidané frázi do slovníku. Týká se to případu, kdy je kódovaný text ukončen dvěma stejnými za sebou následujícími řetězci. Do slovníku i do výstupního souboru zapíšeme řetězec tvořený dekódovanou předchozí frází (stejná jako tato fráze) + prvním znakem této fráze.

Zaveďme následující symboliku. Načítaný kód c ze souboru odpovídá aktuální frázi f ve slovníku, kód c_p odpovídá předchozí frázi f_p. Z těchto frází vytváříme nové fráze F, které přidáváme do slovníku s kódem c. Postup dekomprese vycházející ze [Stan04] můžeme zapsat takto:

1. Do slovníku přidáme všechny znaky z vyskytující se ve vstupním souboru,
2. Načteme první kód c, nalezneme ve slovníku jemu odpovídající frázi f a uložíme ji do výstupního souboru.
3. Dokud nepřečteme všechny znaky vstupního souboru, opakujeme:
   (a) Zapamatujeme si kód předchozí fráze: $c_p = c$.
   (b) Načteme nový kód c ze vstupního souboru. Pokud se kód c (jemu odpovídající fráze f) již ve slovníku vyskytuje, můžeme f okamžitě zapsat do výstupního souboru. Nalezneme frázi f_p k předchozímu kódu c_p a přidáme do slovníku F určené jako
   $$F = f_p + f[0]. \tag{1.17}$$
   (c) Pokud se kód c (jemu odpovídající fráze f) ve slovníku ještě nevyskytuje, přidáme do slovníku i do výstupního souboru novou frázi F
   $$F = f_p + f_p[0]. \tag{1.18}$$

Postup dekomprese pro výstupní data `{1,7,2,6,5,9}` a `{0 1 2 0 2 3 9 11 10 12 15...}` nalezneme v tabulce 1.11. Zvýrazněn je sloupec V obsahující dekomprimovaná data. Dekódování dat je rychlé. Jedinou náročnější operaci představuje prohledávání frází ve slovníku.

```
Výstupní data 172 126 126 126 126 126 1...
```

Naplnění slovníku:							
c_p	Načítáme c	Je c ve slovníku?	f(c)	$f_p(c_p)$	F	Kód C	V
-	-	-	-	-	1	0	-
-	-	-	-	-	7	1	-
-	-	-	-	-	2	2	-
-	-	-	-	-	6	3	-
-	-	-	-	-	5	4	-
-	-	-	-	-	9	5	-
Dekomprese souboru:							
c_p	Načítáme c	Je c ve slovníku?	f(c)	$f_p(c_p)$	Nová fráze $F=f_p+f[0]$	Nový kód C fráze F	V
0	-	-	-	-	-	-	**1**
0	1	Ano	7	1	17	6	**7**
1	2	Ano	2	7	72	7	**2**
2	0	Ano	1	2	21	8	**1**
0	2	Ano	2	1	12	9	**2**
2	3	Ano	6	2	26	10	**6**
3	9	Ano	12	6	61	11	**12**
9	11	Ano	61	12	126	12	**61**
11	10	Ano	26	61	612	13	**26**
10	12	Ano	126	26	261	14	**126**
12	15	-	-	126	1261	15	**1261**

Tabulka 1.11: Dekomprese dat algoritmem LZW.

Účinnost komprese a použití. Algoritmus LZW je asymetrický, méně však než LZ-77. Patří mezi bezeztrátové algoritmy. Velikost slovníku bývá většinou 2^{12} bajtů; tato hodnota ovlivňuje kompresní faktor. U větších slovníků sice dosáhneme lepšího kompresního poměru, doba komprese však výrazně vzroste. Krátké slovníky významně zrychlí dobu komprese, dosažený kompresní poměr je však nižší, v některých případech může dojít i k negativní kompresi. LZW je rychlejší než LZ-77, dosahuje však menšího kompresního poměru. Je součástí řady grafických formátů, např. TIFF, GIF, PS, PDF. Využíván je i v komprimačních programech, např. PKZIP. Algoritmus LZW je, na rozdíl od LZ-77, patentově chráněn.

1.3.4 Transformační metody

Tyto metody využívají geometrické transformace obrazu. Obraz tvořící 3D prostor je aproximován vhodnými matematickými funkcemi, z jejichž koeficientů lze provést zpětnou rekonstrukci obrazu. Nejčastěji se používají následující kompresní algoritmy: JPEG komprese, wavelets komprese, fraktálová komprese. Všechny popsané metody byly navrženy jako ztrátové. Většinou se jedná o metody hybridní kombinující více druhů komprimačních postupů, ať již ztrátových či neztrátových (RLE, LZW, Huffmanovo kódování, ...). V této kapitole se budeme zabývat podrobněji pouze JPEG kompresí, wavelets komprese či fraktálová komprese využívají poměrně robustní matematický aparát, jehož popis by zahrnoval rozsáhlejší výklad.

Faktor komprese q. Úroveň komprimace lze nastavit pomocí faktoru komprese q. Tato hodnota nabývající intervalu $\langle 0, 100\rangle$ ovlivňuje množství informací, které budou při kompresi „zanedbány“, a nepřímo tím ovlivňuje velikost výsledného souboru.

Ztrátové algoritmy využívají faktu, že lidské oko není příliš citlivé (ve srovnáním např. s okem dravců) na barevné a jasové změny (citlivost na změny jasu větší než na změnu barvy) mezi původním a zrekonstruovaným obrazem, a „malých“ změn či odchylek si nevšimne. Ztrátové algoritmy nelze použít pro práci s běžnými dokumenty, jsou určeny pro kompresi rastrových dat, audiodat či videodat. Jejich rozvoj začal v 80. letech 20. století s rozvojem počítačového zpracování obrazu a digitálního videa.

1.3.4.1 JPEG komprese (Joint Photographic Experts Group)

V současné době se jedná o nejpoužívanější techniku komprese statických i pohyblivých rastrů, je využívána ve formátu *.JPG. Algoritmus byl poprvé použit v roce 1991, stal se standardem pro zpracování a uchování obrazových dat.

JPEG komprese dosahuje vysokého kompresního poměru, až 15:1. Je vhodná pro kompresi přirozených rastrů (fotografie), zcela nevhodná pro kompresi rastrů představujících technické výkresy (čarové prvky) či texty. V takovém případě dochází k vizuální degradaci obrazu způsobenou posunem barev a rozmazáním okrajů, která je viditelná pouhým okem. Algoritmus JPEG komprese je symetrický.

JPEG komprese je používána v případech, kdy ostatní komprimační algoritmy nedosahují optimálních výsledků: např. LZW či RLE komprese u rastrů s plynulým barevným přechodem nedosahuje při hledání pravidelně se opakujících vzorů příliš vysoké efektivity. Zvyšováním faktoru komprese dochází k snižování rozdílu barev mezi barvami podobných odstínů, vznikají jednolité barevné oblasti. Další informace o něm lze nalézt v [Žára04], [Peli01].

Princip JPEG komprese. JPEG komprese je založena na faktu, že na malé změny barvy je lidské oko méně citlivé než na malé změny jasu. Nevýznamné změny barev jsou odstraňovány, změny jasu jsou naopak s co největší přesností uchovávány. JPEG komprese představuje poměrně náročný algoritmus, který se skládá z řady dílčích kroků.

Před kompresí dochází k rozdělení obrazu na submatice 8x8. Pokud nejsou šířka nebo výška rastru dělitelné 8, do rastru mohou být doplněny nové řádky či sloupce. Komprese není aplikována na obrázek jako celek, ale na jeho jednotlivé části představované submaticemi, ze kterých se poté skládá výsledný obraz. Postupuje se zpravidla po řadách ve směru z levého horního do pravého dolního rohu rastru. Cílem rozdělení obrazu je snížení ztráty informace při kompresi a dosažení vyššího kompresního poměru.

JPEG komprese kombinuje několik různých postupů pro zvýšení kompresního poměru. Využívá poměrně robustní matematický aparát představovaný diskrétní kosinovou transformací (DCT) aplikovanou na bloky 8x8 pixelů. Algoritmus má „rozumnou“ výpočetní složitost i na průměrně výkonném hardware.

Dělení JPEG komprese. Existují tři typy JPEG komprese, které se liší použitým kompresním poměrem:

1. *Bezztrátová komprese*

   Vhodná spíše pro vědecká data, v praxi je málo často používána. Menší kompresní poměr, cca 2:1.

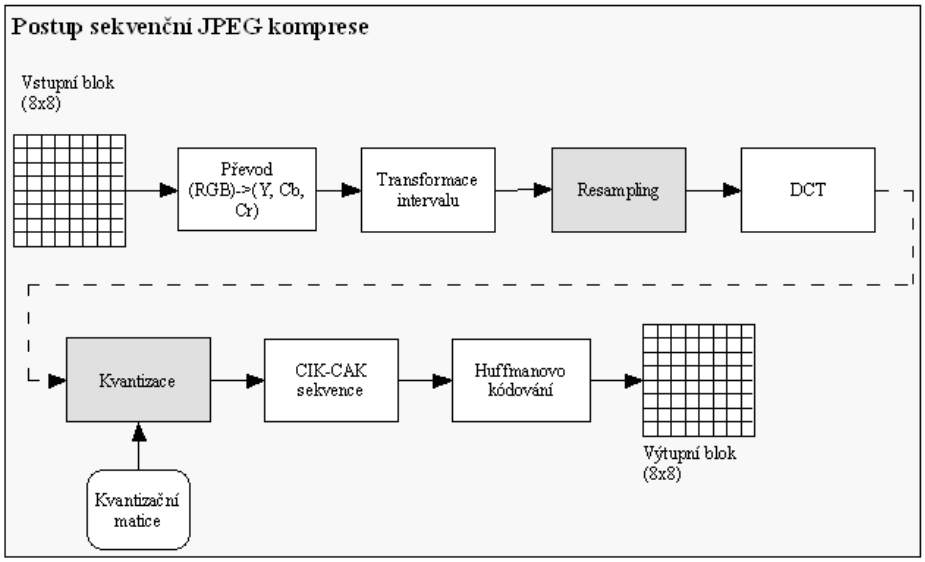
2. *Sekvenční kódování*

   Jediný průchod se zpracováním po blocích. Nejběžnější způsob komprese, kompresní poměr cca 15:1.

3. *Progresivní kódování*

   Komprese rastru po vrstvách, nikoliv po blocích. Umožňuje rychlý náhled na komprimovaná data.

Postup sekvenční JPEG komprese. Postup JPEG komprese se sekvenčním kódováním je znázorněn na obrázku 1.10. Rastrový soubor může mít jiný počet pixelů ve směru osy X nebo Y, matice reprezentující rastr nemusí být regulární. Rastr separujeme na barevné složky R, G, B, komprese bude prováděna po jednotlivých barevných výtažcích, pořadí zpracování složek nehraje roli. Separované barevné složky postupně rozdělujeme na submatice o rozměrech 8x8, které jsou zpracovávány samostatně bez ohledu na okolní submatice. Komprese probíhá ve směru řádků představovaných jednotlivými submaticemi a to z leva do prava. Jednotlivé body komprese stručně popišme.



Obrázek 1.10: Postup sekvenční JPEG komprese, šedě označené kroky jsou ztrátové.

1. **Transformace dat z modelu RGB do modelu** $\mathrm{YC_BC_R}$
   Provádí se z důvodu, abychom při DCT co nejméně poškodili jasovou složku, která bude komprimována méně než obě složky barevné. Z RGB modelu není možné údaje o barevných a jasových složkách získat přímo, musíme ho převést na model $\mathrm{YC_BC_R}$. Vzorec 1.3 lze přepsat do tvaru:

$$\begin{aligned} Y &= 0,2990 \cdot R + 0,5870 \cdot G + 0,1140 \cdot B, \\ C_B &= -0,1687 \cdot R - 0,3313 \cdot G + 0,5000 \cdot B + 128, \\ C_R &= 0,5000 \cdot R - 0,4187 \cdot G - 0,0813 \cdot B + 128. \end{aligned} \tag{1.19}$$

2. **Transformace intervalu**
   Provedeme transformaci intervalu $\langle 0, 255\rangle$ barevných složek Y, C_B, C_R na interval $\langle -255, 255\rangle$, takto získané složky označíme Y', C'_B, C'_R. Tímto krokem docílíme nižší ztráty barevné informace při kompresi.

$$\begin{aligned} Y' &= 2 \cdot Y - 255 \\ C'_B &= 2 \cdot C_\mathrm{B} - 255 \\ C'_R &= 2 \cdot C_\mathrm{R} - 255 \end{aligned} \tag{1.20}$$

3. **Převzorkování (resamplování) rastru**
   Cílem je snížení počtu barevných odstínů rastrového souboru a tím pádem i množství informací obsažených v rastru. Dosáhneme tak vyšší hodnoty kompresního poměru. Tento krok JPEG komprese je ztrátový. Převzorkování provádíme nejčastěji průměrováním submatic o rozměrech 2*2 pixely, 2*1 pixel (sousední pixely) či 3*3 (okolí pixelu).

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

Obrázek 1.11: Ukázka submatice f (8 x 8) představovanou složkou Y.

4. **Diskrétní kosinová transformace (DCT)**
Diskrétní kosinová transformace (Discrete Cosine Transformation) je nejdůležitějším a výpočetně nejnáročnějším prvkem JPEG komprese. Představuje převod prostorových souřadnic (x, y, barva) jednotlivých pixelů do prostorových frekvencí, které jsou vyjádřeny nekonečným množstvím harmonických funkcí cos. DCT převádí obrazový signál závislý na čase posloupnost signálů odlišujících se amplitudou a frekvencí nazývanou frekvenční spektrum.

   Vychází z předpokladu, že rastrové obrazy mají největší množství informací soustředěny v oblastech s nižšími frekvencemi, umožňuje efektivně zakódovat obraz (resp. nejvýznamnější informace v něm obsažené) do poměrně malého množství koeficientů. Existuje několik variant DCT, v našem případě použijeme DCT II. typu.

$$F(u,v) = \frac{1}{4}C(u)\cdot C(v)\left[\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cdot \cos\frac{(2x+1)u\pi}{16}\cdot \cos\frac{(2y+1)v\pi}{16}\right] \tag{1.21}$$

$$\begin{cases} u,v = 0, & C(u) = C(v) = \frac{\sqrt{2}}{2} \\ u,v \neq 0 & C(u) = C(v) = 1 \end{cases}$$

   Hodnota x představuje řádkový index prvků zdrojové submatice, hodnota y sloupcový index zdrojové submatice. Hodnota $f(x,y)$ představuje barevnou informaci obsaženou v x-tém řádku a y-ovém sloupci. Hodnoty u, v tvoří řádkový a sloupcový index v transformované submatice, $F(u,v)$ pak transformovanou hodnotu barevné složky (tj. prostorovou frekvenci) v tomto řádku a sloupci. Výpočet lze realizovat ve čtyřech vnořených cyklech for, viz tab. 12.3, existují však i efektivnější postupy.

Vlastnosti matice F. Po provedení transformace získáme matici, jejíž největší hodnotu má prvek s indexem $F[0][0]$, další prvky s vyššími hodnotami jsou soustředěny v levém horním rohu. Platí následující tvrzení: čím má prvek větší význam, tím je jeho hodnota vyšší; prvky s malou hodnotou mohou být zanedbány.

Prvek $F[0][0]$ je nazýván DC člen, ostatní prvky AC členy. DC člen nejvíce ovlivňuje kvalitu obrazu. Čím jsou hodnoty AC členů menší, tím je jejich vliv na obraz nižší. Některé AC koeficienty s vyššími frekvencemi jsou velmi podobné nebo se liší o malé hodnoty. Pokud by byly stejné, mohli bychom na ně aplikovat efektivněji komprimační algoritmus a dosáhnout vyššího kompresního poměru. O to se budeme snažit v následujícím kroku.

$$F(u,v) = \begin{pmatrix} 1270 & 125 & 20 & 19 & 6 & 5 & -15 & 10 \\ -3 & -80 & 35 & 23 & 40 & 32 & 14 & -8 \\ 104 & -15 & -51 & -32 & -56 & 8 & -4 & -3 \\ -18 & 11 & 11 & 42 & -3 & -11 & -2 & 1 \\ -2 & -15 & -13 & -15 & -8 & 5 & 2 & -2 \\ -16 & 4 & -3 & 11 & -1 & -8 & -4 & 0 \\ 12 & 3 & -1 & -3 & 13 & 3 & 5 & 0 \\ -7 & 8 & 2 & -2 & -5 & -5 & -1 & 1 \end{pmatrix}$$

Obrázek 1.12: Submatice $F(u,v)$ po provedení DCT.

5. **Kvantizace DCT koeficientů**
Představuje nejvíce ztrátovou část JPEG komprese. Nezachovává původní přesnost koeficientů DCT, provádí jejich zaokrouhlení na celá čísla. Cílem kvantizace je vypuštění nepodstatných koeficientů. Jedná se o koeficienty reprezentujících vysoké frekvence, na které není lidské oko příliš citlivé. Výsledkem tohoto procesu bude matice F_{Q}, u které bude řada AC koeficientů stejných (nejen podobných). Kvantizace představuje vydělení matice $F(u,v)$ vhodnou kvantizační maticí $Q(u,v)$. Kvantizační matice více zmenšuje amplitudy vyšších frekvencí než nižších frekvencí.

$$F_Q(u,v) = \frac{F(u,v)}{Q(u,v)} \tag{1.22}$$

Kvantizační matice $Q(u,v)$ bude z důvodu různého stupně komprese barevné a jasové složky jiná pro složku Y i složky C_{B}, C_R. Hodnoty prvků této matice závisejí na nastavení faktoru komprese q. Kvantizační matice se ukládá do JPEG souboru. Faktor komprese bývá zpravidla uváděn v procentech, prvky matice $Q(u,v)$ lze v takovém případě vypočítat ze vztahu:

$$Q(u,v) = \frac{50 \cdot Q(u,v)_{50}}{q}, \tag{1.23}$$

kde q je faktor komprese. Hodnoty $Q(u,v)_{50}$ byly pro složky Y, C_{B},C_R stanoveny empiricky. Všimněme si, že kvantizace je citlivější ke složce Y, prvky matice $F(u,v)$ dělíme menšími hodnotami $Q(u,v)$.

$$Q(u,v)^Y_{50} = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 87 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 26 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix} \qquad Q(u,v)^C_{50} = \begin{pmatrix} 17 & 18 & 24 & 47 & 66 & 99 & 99 & 99 \\ 18 & 21 & 26 & 66 & 99 & 99 & 99 & 99 \\ 24 & 26 & 56 & 99 & 99 & 99 & 99 & 99 \\ 47 & 69 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \\ 99 & 99 & 99 & 99 & 99 & 99 & 99 & 99 \end{pmatrix}$$

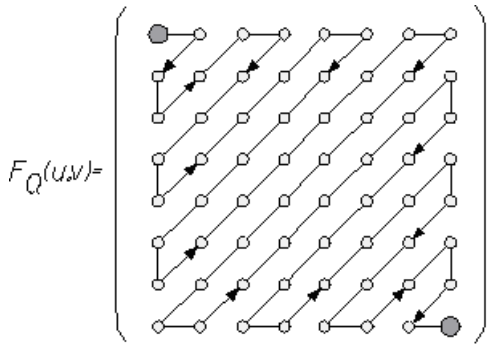
Na následujícím obrázku je znázorněna matice $F_{\mathrm{Q}}(u,v)$ z obr. 1.12 po provedení kvantizace. Faktor komprese byl zvolen: $q = 50$.

$$F_Q(u,v)^Y = \begin{pmatrix} 79 & 11 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & -7 & 3 & 1 & 2 & 1 & 0 & 0 \\ 7 & -1 & -1 & -1 & -1 & 0 & 0 & 0 \\ -1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Obrázek 1.13: Matice $F_Q(u,v)$ po kvantizaci, $q = 50$.

Kvantizovaná matice je maticí řídkou, trojúhelníkovou, u níž zůstal dominantní prvek v levém horním rohu, spousta členů je rovná nule. Při následné kompresi kvantizované matice můžeme dosáhnout vyšších hodnot kompresního poměru než při kompresi matice nekvantizované.

6. **Uspořádání koeficientů do CIK-CAK sekvencí**
   Cílem je vytvořit takovou posloupnost koeficientů, aby mohlo být dosaženo co nejvyšší účinnosti komprese. Koeficienty uspořádáme do struktury, kterou nazýváme CIK-CAK sekvencí (viz obr. 1.14). Počet prvků této posloupnosti odpovídá počtů prvků matice F_Q, prvním prvkem posloupnosti je $F_Q[1][1]$, posledním prvkem $F_Q[8][8]$. Prvky jsou řazeny ve směru diagonál matice F_Q. CIK-CAK uspořádání prvků umožní vytvořit takovou posloupnost, kdy vedle sebe budou umístěny prvky se stejnými hodnotami (zejména koncové prvky posloupnosti představované nulami). Čím je prvek v této posloupnosti dál, tím menší má vliv na kvalitu obrazu.



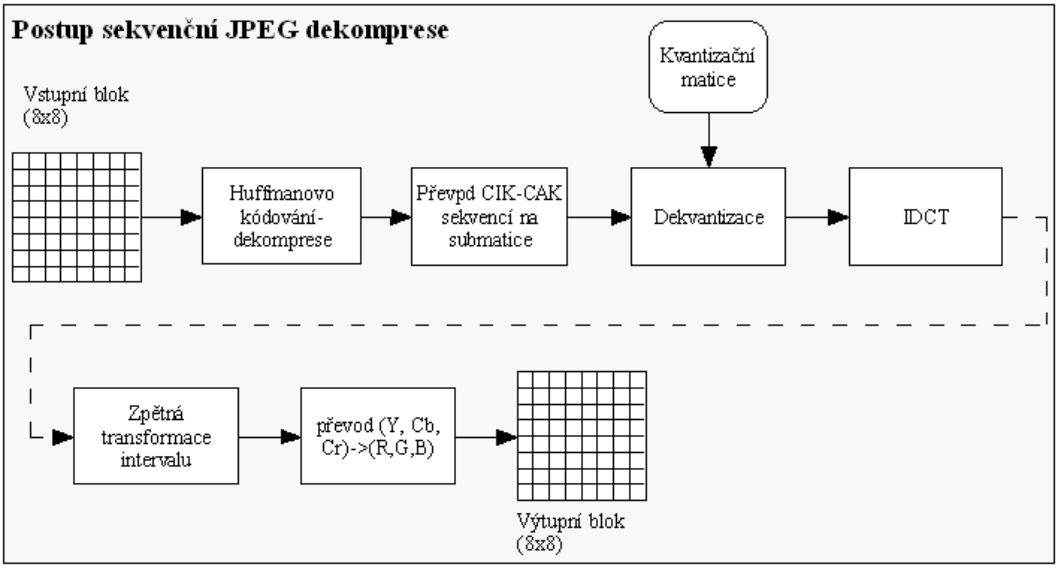
Obrázek 1.14: Uspořádání prvků matice $F_Q(u, v)$ do CIK-CAK sekvencí.

Z obr. 1.14 je patrné, že koeficienty s vyšší hodnotou budou zpracovány dříve než koeficienty s nižší hodnotou. Prvky matice $F_Q(u, v)$ zapsané ve formě CIK-CAK posloupnosti vypadají takto:

```
79 11 0 7 -7 2 1 3 -1 -1 0 1 -3 1 0 0 2 -1 1 -1 -1 0 0 0 1 -1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ...
```

7. **Komprese**
   Prvky CIK-CAK sekvencích jsou komprimovány prostřednictvím Huffmanova kódování. Takto vytvořenou posloupnost lze velmi efektivně komprimovat. Tomu postupu říkáme *entropické kódování*, výsledkem je posloupnost znaků tvořená binárními čísly.

Postup sekvenční JPEG dekomprese
Postup dekomprese je opačný než u komprese. Cílem dekomprese je postupná rekonstrukce obrazu z komprimovaných binárních dat do jednotlivých submatic. Z nich jsou zpětně složeny matice představující barevné složky R, G, B, výsledný obraz vzniká „sečtením“ hodnot prvků těchto matic. Ukázka JPEG dekomprese je znázorněna na obr. 1.15.



Obrázek 1.15: Postup sekvenční JPEG dekomprese.

1. **Dekomprese**
   Dekomprese binárních dat komprimovaných Huffmanovým kódem. Výsledkem je posloupnost dat představující CIK-CAK sekvenci.

2. **Naplnění submatice**
   Prvky CIK-CAK sekvence naplníme submatici $F_Q(u, v)$ o rozměrech (8x8).

3. **Dekvantizace koeficientů**
   Dekvantizace je inverzní operace ke kvantizaci. Na rozdíl od kvantizace není ztrátová, nedochází k zaokrouhlování hodnot vypočtených koeficientů. Dekvantizaci lze zapsat následujícím vztahem:

$$F(u, v) = F_Q(u, v) \cdot Q(u, v). \tag{1.24}$$

4. **Inverzní diskrétní kosinová transformace (IDCT)**
   Převede prostorové frekvence do prostorových souřadnic (x, y, barva) v modelu Y, C_B, C_R, tj. frekvenční spektrum na časové spektrum. Vzhledem k zaokrouhlení při kvantizaci mohou některé z hodnot padnout mimo interval. Musíme proto provést nahrazení takových hodnot krajními hodnotami intervalu (-255 nebo 255). Výsledný vzorec IDCT vypadá takto:

$$f(x, y) = \frac{1}{4}\left[\sum_{u=0}^{7}\sum_{v=0}^{7} C(u) \cdot C(v) F(u, v) \cdot \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16}\right], \tag{1.25}$$

$$\begin{cases} u, v = 0 & C(u) = C(v) = \frac{\sqrt{2}}{2} \\ u, v \neq 0 & C(u) = C(v) = 1 \end{cases}$$

   Hodnota x představuje řádkový index prvků zrekonstruované submatice, hodnota y sloupcový index zrekonstruované submatice. Hodnota $f(x, y)$ představuje výslednou barevnou informaci obsaženou v x-tém řádku a y-ovém sloupci. Hodnoty u, v tvoří řádkový a sloupcový index vstupní submatice, $F(u, v)$ transformovanou hodnotu barevné složky (tj. prostorovou frekvenci) v tomto řádku a sloupci. Postup IDCT je znázorněn v tab. 12.4.

5. **Zpětná transformace intervalu**
   Zpětná transformace intervalu $\langle -255, 255\rangle$ na interval $\langle 0, 255\rangle$. Použijeme následující vzorce:

$$\begin{aligned} Y &= 0,5 \cdot (Y' + 255), \\ C_B &= 0,5 \cdot (C'_B + 255), \\ C_E &= 0,5 \cdot (C'_R + 255). \end{aligned} \tag{1.26}$$

6. **Převod do RGB**
   Převod modelu $YC_{\mathrm{B}}C_{\mathrm{R}}$ do modelu RGB.

Progresivní JPEG komprese. Progresivní JPEG komprese představuje takový způsob kódování obrazu, které umožňuje prakticky okamžité zobrazení náhledu rastru. Využití nalezne při přenášení obrazových dat, typické je použití v prostředí sítě Internet, kdy se náhledy obrázku zobrazují ještě před tím, než byla načtena celá www stránka. Při progresivním kódování neprovádíme kompresi po řádcích jako v případě sekvenční komprese, ale obrázek komprimujeme po vrstvách. Vrstvy se přes sebe překládají jako slidy při promítání. Nejprve zobrazujeme vrstvy s nižší kvalitou obrazu, která se postupně zlepšuje. K zobrazení hrubého náhledu stačí přenést menší množství dat. Používají se dvě metody:

- *Progresivní načítání DCT koeficientů*

  Při této metodě jsou vrstvy představovány DCT koeficienty, jejichž hodnoty jsou následně zpřesňovány. Označíme -li počet vrstev n, představují DCT koeficienty následující posloupnost

$$\{\frac{DCT}{2^{n-1}}, \frac{DCT}{2^{n-2}}, ..., \frac{DCT}{2^0}\}$$

  Pro 4 vrstvy dostáváme posloupnost $\{DCT/8, DCT/4, DCT/2, DCT\}$. Kvalita obrázku se při načítání postupně zlepšuje od hrubého náhledu až po výsledný obraz.

- *Vrstvy jsou představovány intervalem AC koeficientů*

  Nejprve se načítají nejvýznamnější koeficienty, poté koeficienty méně významné. Vrstvy tvoří posloupnost n koeficientů

$$\{DC, \langle AC[2^i - 1], AC[2^{i+1} - 1]\rangle, ..., \langle AC[2^{n-1} - 1], AC[63]\rangle\},\ i \in \langle 1, n-1\rangle$$

  Pro 4 vrstvy dostáváme posloupnost $\{DC, \langle AC[1], AC[2]\rangle, \langle AC[3], AC[6]\rangle, \langle AC[7], AC[63]\rangle\}$, tj. $\{DC\}, \{AC[1], AC[2]\}, \{AC[3], AC[4], AC[5], AC[6]\}, \{AC[7] - AC[63]\}$. Nevýhodou progresivní dekomprese je nutnost opakovaného dekódování celého obrazu, což klade vyšší požadavky na použitý hardware.

Obrázek 1.16: Vliv faktoru komprese na kvalitu přirozeného rastru, vlevo originál, uprostřed $q = 10\%$, vpravo $q = 90\%$. Na pravém obrázku jsou viditelné artefakty komprese se ztrátou detailu.

Vhodnost použití JPEG komprese. Nepřirozené dělení vstupního rastru na submatice (8x8) negativně ovlivňuje výslednou kvalitu dekomprimovaného obrazu. Při vyšším faktoru komprese jsou viditelné „artefakty“ těchto submatic představované pravidelnými plochami pixelů podobných barevných odstínů. Dalším důsledkem je ztráta detailů (rastr je jakoby vyhlazen či zprůměrován). Tento efekt je způsoben výraznějším zaokrouhlením koeficientů $F(u, v)$ při kvantizaci způsobující větší ztrátu informace.

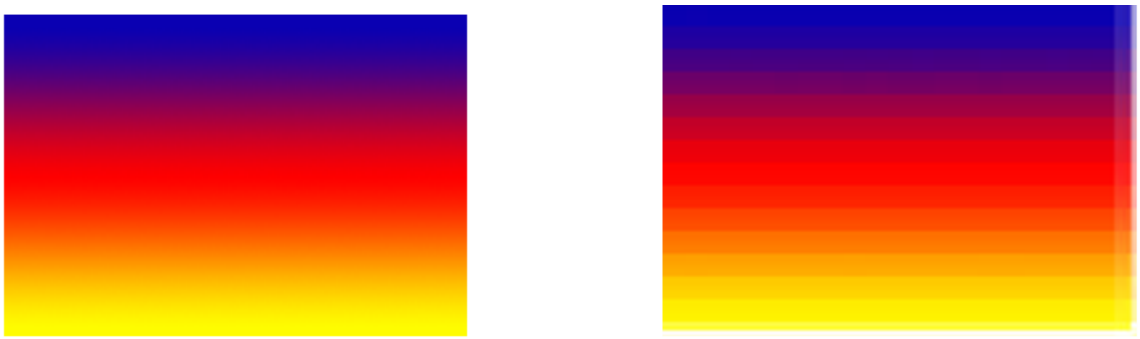
Obrázek 1.17: JPEG komprese rastru v 256 barvách obsahujícího ostré barevné přechody (technický výkres). Všimněme si vizuální degradace obrazu pro $q = 90\%$.

Jak již bylo uvedeno výše, JPEG formát je vhodný pro přirozené rastry představující fotografie. Je poměrně vhodný pro plynulé barevné přechody, avšak nevhodný pro souvislé plochy stejného barevného odstínu (kropenatost), zcela nevhodný pro černobílé rastry či technické výkresy obsahující text nebo vektorovou grafiku, u kterých se vyskytují ostré barevné přechody. V obou posledních případech dochází k takové vizuální degradaci dat, že obrázek již prakticky není možné použít. Projevuje se celkovým silným rozostřením spojeným se ztrátou hran a posunem barev.

Další nevýhodou je ztráta informace v rastrovém souboru při opakovaném ukládání souboru. Nenastane vždy, pouze pokud měníme rozměry rastru či ho ořezáváme tak, že nová hranice nejde po rozhraní bloků. Z výše uvedených důvodů se jako vhodnější alternativa jeví používání „citlivějších“ formátů, např. bezztrátového GIF nebo open source formátu PNG, ze ztrátových algoritmů JPEG2000, fraktální komprese či wavelet transformací.

1.3.4.2 JPEG 2000

Formát JPEG 2000 je nástupcem formátu JPEG. Snaží se odstranit některé jeho nevhodné vlastnosti a stávající vylepšit. Kompresní algoritmus již není založen na DCT, ale používá diskrétní vlnkovou transformaci (DWT). Výsledkem je zhruba o 20-30 procent lepší komprimační faktor a vyšší rychlost komprese. Náhledy jsou díky použité metodice velmi rychlé. Bylo zrušeno omezení velikosti obrazu, které u formátu JPEG činilo 64000x64000 pixelů. JPEG formát byl zaměřen na kompresi přirozených rastrů (fotografie), ve kterých se vyskytovaly spíše plynulé přechody, umělé rastry s ostrými přechody komprimované tímto algoritmem obsahovaly řadu nevhodných artefaktů.



Obrázek 1.18: JPEG komprese plynulého barevného přechodu, $q = 90\%$.

Formát JPEG 2000 byl navržen tak, aby umožňoval provádět efektivní kompresi grafiky s ostrými přechody včetně dokumentů obsahujících vedle sebe grafiku a písmo (pro ně byl původní JPEG formát téměř nepoužitelný). Nový formát by měl podporovat i jiné barevné modely než RGB. Kompresní algoritmus nedělí obrázek na submatice 8x8, pracuje s ním jako s celkem. Algoritmus je víceprůchodový, s větším počtem průchodů klesá komprimační faktor, umožňuje však používat i bezztrátovou kompresi. Do souboru lze dále vkládat různá metadata s údaji o autorských právech či elektronické vodoznaky. Soubor může být komprimován do několika částí s různými kvalitami komprese (celkem 16 zón).

Postup komprese (včetně matematického aparátu) je značně složitý, jeho popis činí cca 200 stran, nebudeme ho proto uvádět. Zájemce ho najde na adrese www.jpeg.org. Tento formát splňuje vysoké kvantitativní i kvalitativní nároky. V oblasti geoinformatiky umožňuje práci se značně rozsáhlými rastry pokrývajícími velká území (družicové snímky). Nejsme omezeni jejich velikostí ani rozlišením, nýbrž pouze dostupným HW vybavením. Podporuje práci s multispektrálními obrazy, které jsou používány pro provádění různých analýz.

1.3.4.3 Fraktálová komprese (PIFS)

Patří mezi ztrátové typy kompresí, používá se pouze pro rastrová data. Jedná se o silně asymetrický algoritmus, doba kódování obrazu je o hodně větší než doba dekódování obrazu. Při fraktální kompresi se vytváří přibližná kopie obrazu, vzniklé chyby jsou jiného druhu než u výše uvedených kompresních algoritmů. Při stejném komprimačním poměru dosahuje lepších výsledků než algoritmus JPEG. Ztráta kvality roste lineárně se zvětšujícím se kompresním faktorem. Jako nejvhodnější se jeví použití fraktální komprese při práci s „přirozenými“ rastry (tj. fotografie).

Základem fraktálové komprese je PIFS (Partitioned Iterated File System). Při aplikaci tohoto algoritmu dochází k opakovanému zmenšování původní situace a hledání soběpodobných míst mezi originálem a zmenšenou kopií. Ve většině přirozených obrázků je však velmi obtížné takovouto podobnost nalézt; úspěšnost je vyšší u umělých obrazů znázorňujících různé geometrické útvary. V praxi se provádí dvojí rozdělení obrázku na:

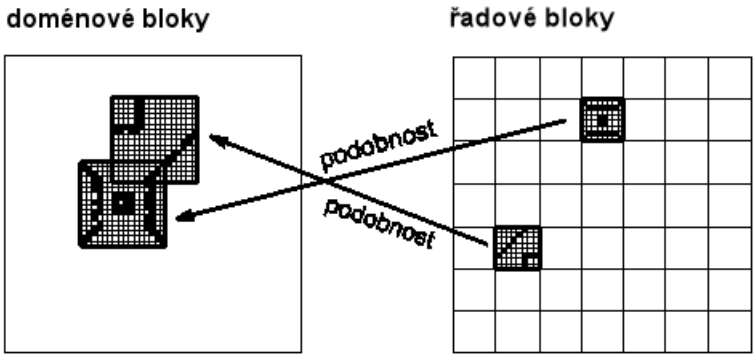
- *Range blocks (řadové bloky)*

  Tvoří je submatice o rozměrech 4x4, 8x8, 16x16, 32x32. Předpokládáme, že budou mít vzhledem k malým rozměrům podobné vzory.

- *Domain blocks (doménové bloky)*

  Bloky obrazu, které jsou co nejvíce podobné řadovým blokům. Jejich rozměry jsou větší, zpravidla tvoří dvojnásobek velikosti řadového bloku (tj. 8x8, 16x16, 32x32, 64x64). Na rozdíl od řadových bloků se doménové bloky mohou překrývat a nemusí pokrýt celý obraz.

Při fraktální kompresi je ke každému řadovému bloku dohledáván nejvíce podobný doménový blok. Aby se zvýšila pravděpodobnost nalezení co nejpodobnějšího bloku, neprovádíme porovnávání řadového a doménového bloku pouze v základní poloze. Doménový blok postupně otáčíme s krokem 90°a



Obrázek 1.19: Nalezení podobných doménových bloků k řadovým blokům,. Rozměr řadového bloku je 8x8 pixelů, rozměr doménového bloku 16x16 pixelů. Doménové bloky se překrývají, mají dvojnásobnou velikost vzhledem k řadovým blokům.

překlápíme ho vodorovně, svisle, podél obou úhlopříček. Pokud nelze k danému řadovému bloku nalézt podobný, je rekurzivně rozdělen na čtyři submatice a tento proces se opakuje. Rozměry submatic se tak postupně zmenšují: 32 – 16 – 8 – 4. Problém lze řešit s použitím rekurze.

Pokud je shoda nalezena již u velkých řadových bloků, dosahuje algoritmus poměrně velkých komprimačních poměrů. Pro každou nalezenou shodu jsou uloženy transformační koeficienty podobnostní transformace: posuny, rotace.

Komprese obrazu. Komprese obrazu je značně výpočetně náročná, zvláště u velkých obrazů je nalezení vzájemně odpovídajících bloků poměrně zdlouhavé. Nelze ho provádět metodou hrubé síly (tj. porovnáváním všech 2x větších submatic), výpočet se urychluje omezováním prohledávané oblasti. Zpravidla se volí okolí řadového bloku vymezené několikanásobkem jeho velikosti nebo prohledávání ve spirále. Začíná na souřadnicích řadového bloku a probíhá po spirále tak dlouho, dokud nenalezneme vhodný doménový blok.

Dekomprese obrazu. Zajímavý je postup dekomprese. Jako základ lze použít libovolný obraz. Na něj jsou iterativně aplikovány uložené transformační koeficienty. Výstupní obraz po první iteraci se stane vstupním obrazem pro provedení druhé iterace. Jako vstupní obraz se používá nejčastěji jednobarevná plocha šedé barvy, výpočet pak konverguje velmi rychle, postačuje cca 3-5 iterací.

JPEG komprese dosahuje maximálního kompresního poměru cca 50:1, fraktální komprese až 300:1. Rastry komprimované pomocí JPEG komprese jsou v tomto stupni komprese vzhledem k množství šumu a ztrátě detailu obtížně použitelné, restry komprimované fraktální transformací jsou mnohem kvalitnější.