Technická zpráva

Akademie věd České republiky
Ústav teorie informace a automatizace AV ČR, v.v.i.

Kompresní algoritmy a jejich implementace

B. Kovář, J. Schier, RIPAC projekt
Kovar@utia.cas.cz, http://www.ripac.cz

Obsah

Revize

Revize	Datum	Autor	Popis změn v dokumentu
0	10.11.2007	B.K.	Vytvoření dokumentu
1			
2			

1. Úvod

V tomto dokumentu jsou popsány základní metody komprese obrazu. Výběr vhodné kompresní techniky je v algoritmech zpracování obrazu klíčový. Barevný obraz v PAL rozlišení představuje datový tok přibližně 30MB/s. Přenosová kapacita (zvláště ta bezdrátová) je i na velmi krátkou vzdálenost v řádu desítek metrů několikrát menší. Datový tok je tedy potřeba snížit. Toho můžeme dosáhnout:

1. snížením rozlišení obrazu (při rozlišení PAL/2 klesne datový tok 4x),
2. kompresí obrazu,
3. snížením rozlišení a kompresí.

Snížením rozlišení ovšem přicházíme i o detaily, které jsou nepostradatelné například pro systémy rozpoznávání obrazu, případně pro stereo-vidění. Z tohoto důvodu se zdá komprese obrazu vhodnější řešení. V této zprávě jsou uvedeny základní metody komprese obrazu a na experimentech je demonstrována jejich vhodnost pro využití v projektu RIPAC.

2. Metody komprese obrazu

3. Redundance a irelevance

Redundance a *irelevance* vyjadřují všeobecně nadbytečnou informaci v signálu. To znamená, ze signál, který obsahuje tyto informace, se nám jeví stejně i po odstranění těchto informací. V čem je rozdíl mezi redundantní a irelevantní složkou? Rozdíl spočívá v objektivním resp. subjektivním pohledu na signál. *Redundantní* složka totiž představuje tu část informace, kterou když odstraníme ze signálu, ze zbývající části můžeme rekonstruovat původní signál bez zkreslení. Například mějme signál, který je reprezentován následující sekvencí vzorku $s(n)$ *:*

$$s(n) = \{2,2,2,2,2,2,2,2,2,2,6,6,6,6,6,7,7,7\}.$$

Tuto sekvenci můžeme jednoznačně popsat i pomocí kratšího vyjádření. Rozdělme sekvenci $s(n)$ na tři subsekvence takovým způsobem, aby každá subsekvence obsahovala jen vzorky stejných hodnot:

$$\{2,2,2,2,2,2,2,2,2,2\},\{6,6,6,6,6\},\{7,7,7\}.$$

Potom každou tuto subsekvenci lze vyjádřit dvojicí, která definuje hodnotu vzorku a počet opakování v ní. Na základě toho signál $s(n)$ může být jednoznačně vyjádřen pomocí tří dvojic takto:

$$\{2,10\}\{6,5\}\{7,3\}.$$

Z tohoto kratšího vyjádření na základě znalosti jeho struktury (první číslo v dvojici znamená hodnotu a druhé číslo počet vzorků) je možné rekonstruovat sekvenci $s(n)$, aniž bychom ztratili nějakou informaci.

Irelevantní informace je ta část informace, jejíž nepřítomnost je nepostřehnutelná v signálu. Signál původní a signál s odstraněnou irelevantní částí se nám jeví stejně, i když stejné nejsou. Například mějme sekvenci $f(n)$, která reprezentuje hodnoty jasu v okolí bodu (i, j) černobílého obrazu:

$$f(n) = \{7,6,6,6,6,5,5,6,7\}.$$

Průměrná hodnota jasu v této sekvenci je 6. Pokud přepíšeme sekvenci $f(n)$ do tvaru $g(n)$:

$$g(n) = \{6,6,6,6,6,6,6,6,6\},$$

Můžeme ji jednoduše zkomprimovat jako $\{6,9\}$. Jak vidíme, šedotónové vyjádření sekvence $f(n)$ se od sekvence $g(n)$ vizuálně nijak neliší. Sekvenci $g(n)$ ale můžeme jednoduše komprimovat ve

tvaru $\{6,9\}$, což u sekvence $f(n)$ nebylo možné. Odstraněné malé změny intenzity ze sekvence $f(n)$ už nejsou rekonstruovatelné ze sekvence $g(n)$. Z vizuálního hlediska se však tyto dvě sekvence jeví stejně, i když stejné nejsou. Tyto dva druhy nadbytečných informací umožňují konstrukci dvou typů komprese – *bezeztrátové* a *ztrátové*.

4. Bezeztrátová komprese obrazu

4.1 Run Length kódování

Run Length Encoding (RLE) je jednou z nejjednodušších metod bezeztrátové komprese. Opakující se hodnoty jsou nahrazeny strukturou, která udává danou hodnotu a počet opakování. Každá taková sekvence je zakódována pomocí tří bytů. Například sekvenci

$$f(n) = \{17,16,6,56,3,3,3,3,7\}$$

můžeme pomocí RLE zakódovat takto:

$$g(n) = \{17,16,6,56,\mathbf{0},\mathbf{3},\mathbf{4},7\},$$

Kde **0** představuje uvozující znak (marker) a hodnota **3** je **4**-krát zopakována. Použitím tohoto kódu je sekvence $f(n)$ zkrácena o 1byte. Pro svou jednoduchost je tento typ komprese v obecných problémech velmi málo efektivní. V některých speciálních případech (například jako součást JPEG komprese) je velmi užitečný. Jako marker je určen znak, který se v sekvenci nevyskytuje, případně je nejméně častý.

4.2 Huffmanovo kódování

Huffmanovo kódování je jednou z nejlepších metod bezeztrátové komprese. Každý symbol vyskytující se v datovém souboru je nahrazen alternativní bitovou reprezentací. Počet bitů, který je pro popis znaku použit, záleží na frekvenci výskytu daného znaku v souboru. Často se opakující znak je zakódován pomocí pár bitů, znak, který se vyskytuje pouze několikrát je popsán podstatně delší bitovou sekvencí. Klíčovým problémem je tedy zjistit četnosti výskytu jednotlivých znaků v nekomprimovaném datovém souboru. To můžeme zjistit například použijeme-li histogram. Získaný histogram je poté rekurzivně rozdělován na dvě poloviny a překódován do podoby binárního stromu. Každá větev binárního stromu by měla mít přibližně stejnou váhu. Váha je v tomto případě definována jako:

$$W_i = \sum_{k=1}^{N} x(k)$$

kde, *i* je index větve binárního stromu, *N* počet symbolů na *i*-té větvi a *x(k)* četnost *k*-tého znaku. Princip Huffmanova kódování můžeme demonstrovat takto:

32	22	22	43	49	22	22	17	48	43

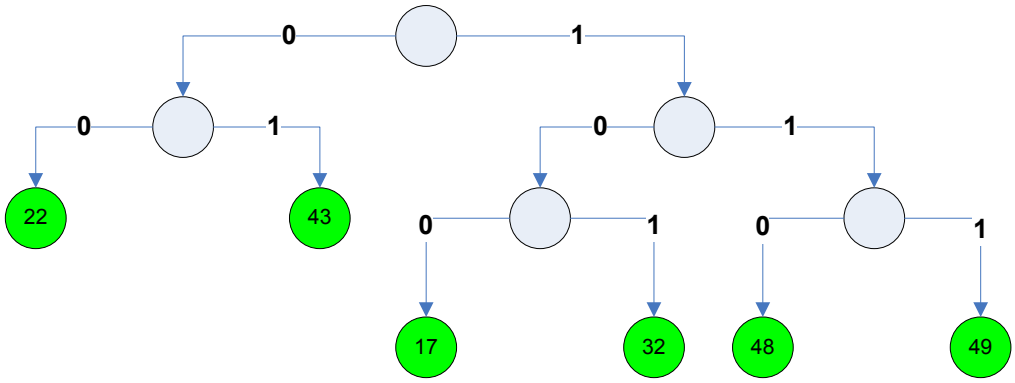
Tabulka 1: Nekomprimovaný datový stream, vstup Huffmanova kodéru.

Četnosti jednotlivých prvků a odpovídající binární reprezentace tedy je:

Symbol	Četnost	Kód
22	4	00
43	2	01
17	1	100

32	1	101
48	1	110
49	1	111

Tabulka 2: Znaky, jejich četnosti a na základě binárního stromu přiřazené kódy.



Obrázek 1: Binární strom odpovídající zadané datové sekvenci

101	00	00	01	111	00	00	100	110	01

Tabulka 3: Komprimovaná datová sekvence

4.3 Rice kódování

Rice kódování je velmi účinné na 16 nebo 32 bitová data s velkou dynamikou (tzn. hodnoty dat jsou malá nebo naopak velká čísla). Typicky se může jednat o audio data, případně vhodně předzpracovaná obrazová data. Přestože Huffmanova komprese je pro tento typ dat také vhodná, z praktických důvodů se nepoužívá. Konstrukce binárního stromu pro 32bitová data vyžaduje 16GB operační paměti. Základní princip Rice kódování je stejný jako u Huffmana. Uložit co nejvíce znaků použitím co nejméně bitů. U tohoto typu kódování se však na rozdíl od Huffmana neprovádí statistická analýza vstupního souboru. Předpokládá se pouze, že vzhledem k velké dynamice ve vstupních datech, bude výskyt malých, resp. Velkých hodnot častější.

5. Ztrátová komprese

5.1 JPEG komprese

Ztrátová komprese JPEG se dá rozdělit do několika kroků. Nejdříve se provádí transformace do barevného modelu YUV, následně se provede diskrétní kosinová transformace, pak kvantizace a nakonec bezeztrátová komprese. Transformace do modelu YUV je jednoduchá lineární transformace, v modelu YUV pak složka Y odpovídá jasu (luminanci) a složky U a V odpovídají modré nebo červené složce barvy (chrominance). Chrominační údaje se pak jako nedůležité podvzorkují na nižší rozlišení.

$$Y = 0.299R + 0.587G + 0.114B$$
$$C_b = 0.1687R - 0.3313G + 0.5B$$
$$C_r = 0.5R - 0.4187G - 0.0813B$$

Díky výpočtu diskrétní kosinové transformace se nezávisle každý blok 8×8 pixelů převede na 8×8 amplitud různých 'dvojrozměrných kosinových vln', které po složení mají v těch 8×8 bodech stejnou hodnotu jako původní blok. Tato transformace sama o sobě není ztrátová ani nijak nezmenší množství přenášených údajů, ale údaje transformuje do podoby, která je nutná pro další krok. Pokud totiž po výsledné matici 8×8 se pohybujeme metodou *'zig-zag'* od vrcholu (0, 0) do vrcholu (7, 7), tak postupně klesá (u běžných fotografií) důležitost těchto údajů.

Na každou takovou matici se pak uplatňuje kvantizace. Podle zadané 'kvality' či 'ztrátovosti' existuje pro celý obrázek jedna tzv. kvantizační matice a jednotlivé matice získané v předchozím kroku se pak touto maticí vydělí po složkách (a zaokrouhlí). Tím se zvláště v pravém dolním rohu objeví mnoho nul (tam má totiž kvantizační matice poměrně velké koeficienty). No a taková posloupnost dat se již velice úspěšně zkomprimuje nějakou neztrátovou kompresí, jako třeba Huffmanovým kódováním nebo aritmetickou kompresí. Dekomprese pak probíhá zcela symetricky.

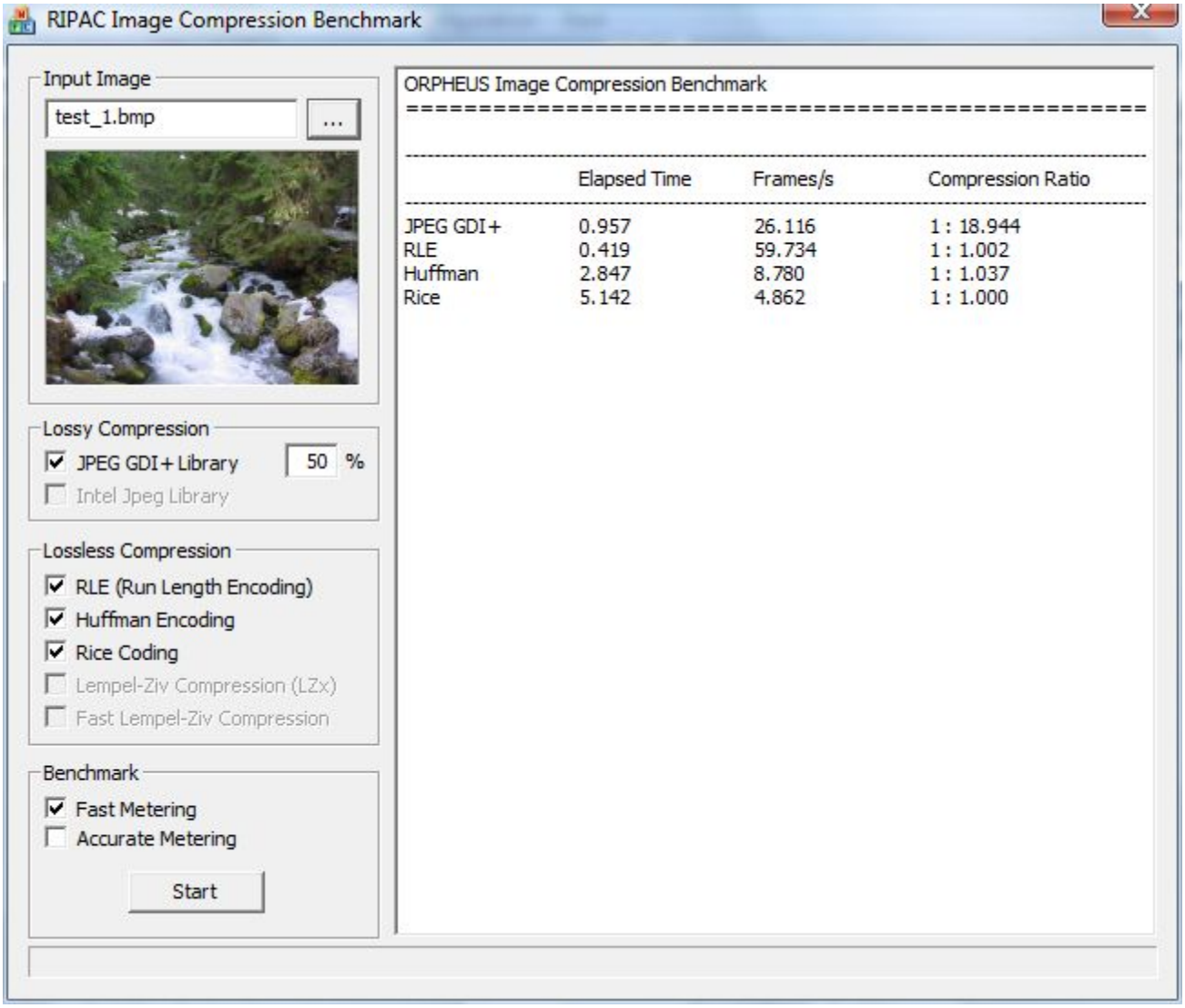
6. Implementace

Pro výběr vhodné kompresní metody, která by vyhovovala definovaným parametrům projektu RIPAC, byla vytvořena aplikace umožňující porovnání dostupných komprimačních algoritmů z hlediska výsledného kompresního poměru, rychlosti komprimace a dekomprimace a tím i algoritmické náročnosti. Zkoumán byl i vliv obsahu snímané scény na jednotlivé kompresní algoritmy a výsledné kompresní poměry.

Aplikace *CBench* byla naprogramována v jazyce C++ v prostředí Microsoft Visual Studio .NET 2003. Kromě standardních Win32 knihoven MFC byly použity následující volně dostupné knihovny:

1. GDI+ (je součástí MSVS .NET 2003),
2. Independent JPEG Group's library,
3. BCL v. 1.1.1.

Aplikace byla optimalizována pro rychlost a nejnovější řadu procesorů Intel. Výsledků, uváděných v kapitole 4, bylo dosaženo na notebooku s operačním systémem Windows XP SP2, procesorem Intel Pentium-M 1.7GHz a 512MB operační paměti.



Obrázek 2 Aplikace CBench

7. Výsledky a experimenty

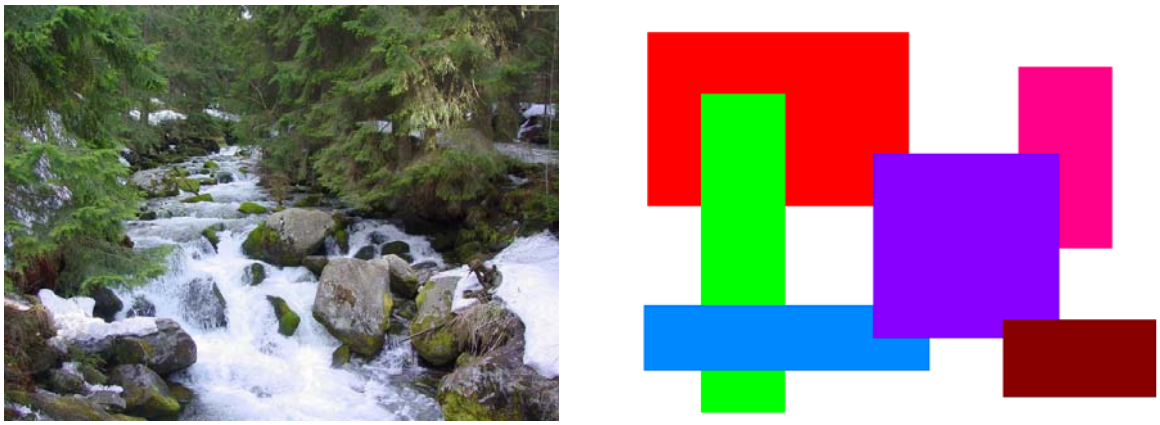
V této kapitole uvádíme výsledky experimentů, které byly prováděny během vývoje aplikace CBench a při výběru vhodné kompresní techniky. Při experimentech jsme se snažily zkoumat zejména:

1. rychlost komprese,
2. algoritmickou náročnost použitého algoritmu,
3. výsledný kompresní poměr,
4. vliv snímané scény na kompresní poměr,
5. subjektivní kvalitu výsledku.

Vyhodnocovány byly jak statické snímky tak i video sekvence snímané barevnou FireWire kamerou s PAL rozlišením. V tomto dokumentu je ukázána zejména univerzálnost JPEG komprese pro různé typy obrazu a naopak nevhodnost nejjednodušších bezeztrátových technik pro reálné (tzn. ne uměle vytvořené obrazy). Všechny experimenty byly provedeny desetkrát, uvedené výsledky jsou průměrné hodnoty.

7.1 Vliv obsahu obrazu na výsledky komprese

Tento experiment měl demonstrovat citlivost jednotlivých algoritmů na obsah komprimovaného obrazu. Zkoumány byly dosahované kompresní poměry a subjektivní degradace obrazu při zvyšování kompresního poměru.



(a) (b)

Obrázek 3: V tomto dokumentu byly použity tyto dva testovací obrázky. Obrázek (a) představuje reálný obraz s velkým množstvím detailů, obrázek (b) je synteticky vytvořený.

	Obrázek A		Obrázek B	
Kompresní metoda	*kompresní poměr*	*čas*	*kompresní poměr*	*čas*
GDI+ JPEG	18,944	0,982	76,996	0,741
RLE	1,002	0,371	1,810	0,405
Huffman	1,034	2,074	5,371	0,761
Rice	1,000	4,399	1,683	2,043

Tabulka 4. Vliv obsahu obrazu na použité kompresní metody.

7.2 Vliv kvality JPEG na stupeň komprese

Tento experiment má demonstrovat vliv nastavení kvality JPEG na výsledný kompresní poměr a potřebný čas pro kódování. Každý experiment byl proveden 25x, uvedené časy jsou průměrné. Výsledek je ovlivněn multitaskingem systému Windows a časem potřebným pro zápis výsledku (s rostoucí kvalitou roste i velikost výsledného JPEG souboru).

JPEG Kvalita	kompresní poměr	čas	snímků/s
10%	60,929	0,03716	26,911
20%	36,133	0,03716	26,904
30%	26,857	0,03976	25,148
40%	22,148	0,03776	26,494
50%	18,944	0,03872	25,823
60%	16,361	0,04008	24,945
70%	13,516	0,04104	24,366
80%	10,584	0,04648	21,506
90%	7,159	0,04848	20,621
100%	2,763	0,06952	14,387

Tabulka 5: Vliv kvality JPEG na kompresní poměr a čas komprese.

8. Závěr

Experimenty bylo potvrzeno, že ty nejjednodušší bezeztrátové kompresní metody nejsou příliš vhodné pro reálná obrazová data. Akceptovatelných kompresních poměrů lze dosáhnout pouze pro uměle vytvořené obrazy. Z dostupných ztrátových komprimačních technik, ve všech kategoriích, nejlepších výsledků dosahoval JPEG. Výhodou je nejen kompletní dokumentace, OpenSource zdrojové kódy, ale i možnost přizpůsobit stupeň komprese (a tím i kvalitu výsledku) okamžitému stavu přenosového kanálu.

Obsah a popis přiloženého balíku

```
cdrom    - Dll
         - Doc
         - Img
         - Lib
              CBench.exe
```

Aplikaci je možné spouštět přímo z přiloženého CD. V případě, že na počítači nejsou instalovány vývojové nástroje firmy Microsoft, je nutné adresářovou strukturu překopírovat na pevný disk počítače a do adresáře s programem CBench.exe (případně Windows/System32) p5esunout knihovny z adresare Dll.