VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

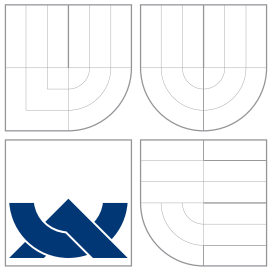
VIDEOKODEK - KOMPRESE VIDEOSEKVENCÍ

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

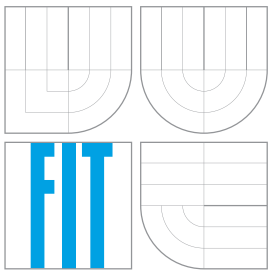
AUTOR PRÁCE
AUTHOR

JAN VANČURA

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÝCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER SYSTEMS

VIDEOKODEK - KOMPRESE VIDEOSEKVENCÍ

VIDEOCODEC - VIDEOSEQUENCE COMPRESSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — JAN VANČURA
AUTHOR

VEDOUCÍ PRÁCE — Ing. STANISLAV SUMEC
SUPERVISOR

BRNO 2007

Zadání bakalářské práce/5854/2006/xvancu08

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Vančura Jan**
Obor: Informační technologie
Téma: **Videokodek - komprese videosekvencí**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte techniky používané pro kompresi obrazu a videosekvencí. Zaměřte se na ztrátovou kompresi.
2. Navrhněte postup komprese videosekvencí s použitím vybraných technik.
3. Implementujte videokodek pro kompresi a dekompresi videosekvencí pracující na základě algoritmu navrženého v bodu 2.
4. Integrujte videokodek do prostředí operačního systému Microsoft Windows formou DirectShow filtru.
5. Proveďte porovnání výsledků s běžně používanými kodeky využívajících ztrátovou kompresi.
6. Diskutujte dosažené výsledky a možnosti případného pokračovaní práce.

Literatura:

- dle pokynů vedoucího

Při obhajobě semestrální části projektu je požadováno:

- Body 1. a 2.

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Sumec Stanislav, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. **Pan**

Jméno a příjmení: **Jan Vančura**
Id studenta: 84117
Bytem: Heyrovského 859/6, 674 01 Třebíč
Narozen: 30. 03. 1984, Brno
(dále jen "autor")

a

2. **Vysoké učení technické v Brně**

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Videokodek - komprese videosekvencí
Vedoucí/školitel VŠKP: Sumec Stanislav, Ing., Ph.D.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Tato práce se zabývá návrhem a implementací aplikace pro kompresi videosekvencí. Aplikace je implementována za použití programovacího jazyka C++ pod Microsoft Visual Studiem 2005. Využívá knihovny Video for Windows a DirectShow. Má za cíl snížení velikosti datového toku videa. Popisuje ztrátové a bezeztrátové metody komprese.

Klíčová slova

video komprese, C++, DCT, Exp-Golomb, Golomb-Rice, Huffman, Video for Windows, DirectShow

Abstract

This thesis is concerned with design and implementation of application for video compression. Application is implemented with programming language C++ in Microsoft Visual Studio 2005 by using library Video for Windows and DirectShow. It is focused to decrease size of video data stream. It describes lossy and lossless data compression.

Keywords

video compression, C++, DCT, Exp-Golomb, Golomb-Rice, Huffman, Video for Windows, DirectShow

Citace

Jan Vančura: Videokodek - komprese videosekvencí, bakalářská práce, Brno, FIT VUT v Brně, 2007

Videokodek - komprese videosekvencí

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Stanislava Sumce. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....

Jan Vančura

14. května 2007

Poděkování

Chtěl bych poděkovat Ing. Stanislavu Sumcovi za jeho odbornou pomoc při psaní této práce.

Obsah

Kapitola 1

Úvod

Cílem této bakalářské práce bude vytvořit jednoduchý *videokodek*. Pojem videokodek vznikl složením několika slov. Slovo kodek označuje počítačový program, který dokáže transformovat datový proud do zakódované formy. Tento proces se nazývá kódování. Častěji se u kodeků používá opačný proces označovaný jako dekódování. Slovo **kodek** je tak spojení počátečních písmen dvou slov **kó**dování a **dek**ódovaní. Přidáním slova **video** vzniká aplikace provádějící kompresi a dekompresi videosekvencí. Kompresí videa se pak rozumí použití různých metod komprese za účelem snížení objemu dat.

Kapitola 2 se zabývá historickým vývojem komprese v oblasti multimedií. V kapitolách 3 a 4 jsou popsány metody ztrátové a bezeztrátové komprimace, které jsou využívány při kompresi videa. Návrh a implementace aplikace se nachází v kapitole 5. S výslednou aplikací jsou provedeny testy, které jsou obsahem kapitoly 6. V závěrečné kapitole 7 je zhodnocena celková aplikace a jsou nabídnuty další možnosti rozšíření.

Kapitola 2

Historie

Již od pradávna se člověk setkával se situacemi, kdy byly jeho možnosti omezené. Pračlověk odcházející na lov se vracel pouze s kořistí, kterou sám unesl. Všechny tyto situace byly podnětem k lidské činnosti. A tak člověk vynalez kolo a vznikl vůz. Tím se jeho schopnost transportu věcí výrazně zlepšila. Postupem času se však počet přepravovaných věcí zvětšoval a člověk potřeboval odlehčit (zkomprimovat). Rozhodl se tedy, že věci na voze přeskládá tak, aby ušetřil nějaké místo (bezeztrátová komprimace). Popřípadě zbytečné věci nechal doma (ztrátová komprimace).

Příchodem počítačů začala komprimace nabírat na důležitosti. Začaly se objevovat různé komprimační algoritmy, které se staly základem úspěchu mnoha programů. Komprimační algoritmy lze rozdělit do dvou základních kategorií. *Bezeztrátová komprimace* při které se zachovávají všechna data a data jsou po dekódování zcela totožná s původními. Naopak při ztrátové komprimaci jsou některé informace nenávratně ztraceny. *Ztrátová komprimace* je mnohem účinnější než bezeztrátová. Nevýhodou je ale určité zkreslení oproti originálu. Ztrátová komprimace využívá hlavně pro kompresi dat určených k smyslovému vnímání (obraz, zvuk, video). Právě ztrátová komprimace zaznamenala v posledních letech největší rozvoj.

V roce 1986 je ustanovena skupina Joint Photographic Experts Group (zkráceně **JPEG** [13]) dohlížející na nový standard obrazového formátu. V roce 1992 došlo k jeho vydání na veřejnost. Standard JPEG specifikuje způsob komprimace obrazových dat a způsob jejich dekomprimace. Zanedlouho se stal velmi oblíbeným a prosadil se do mnoha odvětví digitální techniky. I současnosti je jedním z nejrozšířenějším obrazovým formátem. Obrazy využívající standard JPEG mají nejběžnější souborovou příponu .jpg.

Příchodem formátu JPEG dochází v počítačové technice k rozvoji multimediálních služeb. V 90. letech je tento rozvoj ještě více vystupňován příchodem internetu a novými instrukčními sadami (MMX, SSE), které v procesorech zvyšující výkon práce s multimédii. Objevuje se mnoho programů pro práci s digitální technikou (fotky, videa, zvuk). Z JPEG se vyvinul video formát MJPEG, který jednotlivé snímky komprimuje jako JPEG. Snímky formátu MJPEG nejsou mezi sebou datově propojeny, a tak je tento formát vhodný pro

střih videa. Asi největší vzestup zaznamenal hudební formát MP3. Na trhu se objevují scannery, digitální fotoaparáty, videokamery a mnoho dalších. Vše co bylo dříve drahou záležitostí, je stále dostupnější každému. Ovšem takovýto rozmach technologií potřebuje odborný dohled. Proto vznikají skupiny, které dohlížejí na standardy formátů. Nejznámější skupinou dohlížející na formáty videa se jmenuje MPEG.

2.1 MPEG

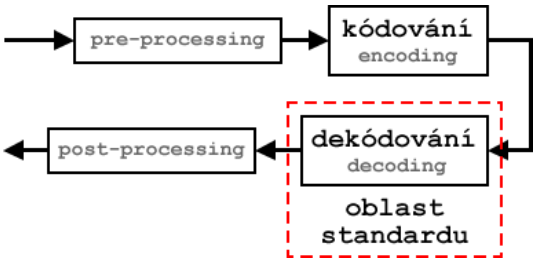
V květnu roku 1988 se konalo první setkaní MPEG (Moving Picture Experts Group). Cílem práce této skupiny bylo standardizovat metody komprese videosignálu a vytvořit otevřenou a efektivní kompresi. Skupina zavedla mnoho úspěšných standardů, které jsou dodnes široce používány. Klade důraz na strukturu uložení dat komprimace, tak aby bylo možné podle standardu navrhovat např. hardwarové přehrávače, řídící jednotky v digitálních videokamerách a další. Standardy skupiny MPEG dobře dokumentují, jak se komprese videa postupem času vyvíjela.

Prvním standardem byl formát **MPEG-1**. Byl dokončen v roce 1991 a jako norma přijat v roce 1992. Byl navržen pro práci s videem o rozlišení 352x288 bodů a 25 snímků/s při datovém toku 1500kbit/s. MPEG-1 komprese používá ke kompresi videa I, P a B snímky, které se používají v kompresi videa dodnes. I-snímky (Intra Pictures) jsou snímky klíčové, jsou komprimovány obdobně jako MJPEG, ale navíc s možností komprimovat jednotlivé části obrazu odlišným stupněm komprese. P-snímky (Predicted Pictures) jsou kódovány s ohledem na nejbližší předchozí I-snímek nebo P-snímek. B-snímky (Bidirectional Pictures) jsou pak dopočítávané jako rozdílové snímky mezi nejbližším předchozím I-snímkem nebo P-snímkem a nejbližším následujícím I nebo P-snímkem. Celá sekvence snímků (od jednoho I- po další I-snímek) se pak nazývá GOP (Group of Pictures). Standardní MPEG stream pro VCD, SVCD a DVD používá pořadí IBBPBBPBBPBBPBBPBB. Vzdálenosti mezi I-snímky se mohou lišit v závislosti na změně scény a další faktorech. Velikost GOP vede především ke zlepšení kvality videa. Poměr I-, P- a B-snímků lze většinou nastavit, záleží na implementaci kompresoru. Z pohledu obsazeného místa I-snímky zabírají nejvíce, po nich jsou P-snímky a úplně nejméně místa zabírají snímky B. Způsob komprese navržený v MPEG-1 se stal výchozím základem dalších standardů. Parametry komprese MPEG-1 jsou srovnatelné s analogovým formátem VHS.

Další úspěšný standard **MPEG-2** dovoluje pracovat s prokládaným videem a obsahuje podporu vysokých rozlišení. Oproti předchůdci zatěžuje více procesor právě kvůli vysokému rozlišení. Standard **MPEG-3** byl určen pro HDTV (televize s vysokým rozlišením). Jeho vývoj byl ale zastaven, protože pro požadavky HDTV plně postačuje formát MPEG-2.

Standard **MPEG-4** nemá přesně definovanou kompresi a komprimační algoritmy, ale pouze jde o množinu parametrů a vlastností, které musí kompresor splňovat, aby byl MPEG-4 kompatibilní. Na rozdíl od předchozích standardů nedefinuje oba způsoby komprimace. Vliv standardu se vztahuje pouze na proces dekódování (viz obrázek 2.1). Důvodem

tohoto postoje je existence mnoha implementací. Standard MPEG-4 se nezabývá pouze kompresí videa, ale definuje globální celek, který s videem souvisí. Standard je rozdělen do několika částí. Mezi nejpoužívanější patří část 2 označována jako MPEG-4 ASP (Advanced Simple Profile). Popisuje kompresi videa, která je dnes známa pod implementací kodeků DivX a XviD. Následující část 3 MPEG-4 AAC (Advanced Audio Coding) nese pokročilou kompresi zvuku. MPEG-4 AVC (Advanced Video Coding) v 10. části se zaměřuje na pokročilou kompresi videa s použitím nejnovějších technik. Příkladem implementace je x264, dosahující o 30 % lepší kompresibility oproti MPEG-4 ASP.



Obrázek 2.1: Oblast vlivu standardu

2.2 Současnost

Na přelomu století skupina JPEG začala prosazovat nový obrazový formát pojmenovaný **JPEG2000**. Využívá vlnkovou transformaci oproti diskrétní kosinové transformaci použité ve formátu JPEG. Tím je zajištěn o 30 % lepší kompresní poměr při zachování kvality. JPEG2000 má nahradit již zastaralý JPEG, ale z důvodů rozšířenosti JPEG a patentovému ochraně JPEG2000 se tak ještě nestalo. Zvyšování výkonu počítačů přináší nové metody komprese, které se dříve nemohly efektivně použít. Příkladem může být standard MPEG-4 část 10, kde se popisují nové způsoby komprimace videa.

Od počátku 21. století se také začínají prosazovat počítače do obývacích pokojů, kde se na multimediální služby klade největší důraz. To vše zvyšuje v současnosti důležitost zpracování videa, hudby a obrazu.

Kapitola 3

Bezeztrátová komprimace

Bezeztrátová komprimace dovoluje přesnou zpětnou rekonstrukci komprimovaných dat. Používá se všude tam, kde je důležité, aby originální data byla totožná s daty po dekompresi komprimovaného souboru - např. komprese textů nebo komprese čehokoli, kde je nepřípustná i sebemenší ztráta informace. Existují 4 základní druhy metody bezeztrátové komprimace:

- **Transformace** - Modifikují data tak, aby se dala lépe zkomprimovat. Musí existovat transformace inverzní.
- **Slovníkové metody** - Vytvářejí v průběhu komprimace slovník na základě dat již zkomprimovaných. Pokud jsou data nalezena ve slovníku, algoritmus zapíše pozici dat ve slovníku místo samotných dat.
- **Statistické metody** - Snaží se určitým způsobem předvídat jaké znaky budou v souboru dat následovat. Pro znaky s vyšší pravděpodobností výskytu vyhradí algoritmus kratší informaci pro jejich zapsání. Statistické metody dále dělíme na metody se statickým modelem a metody s adaptivním modelem. Metody se statickým modelem vytvoří před komprimací dat určitý model a podle něho zkomprimují celý soubor dat, zatímco metody s adaptivním modelem průběžně model aktualizují. Obecně se dá říci, že metody se statickým modelem bývají dvou-průchodové a metody s adaptivním modelem jedno-průchodové.
- **Ostatní metody**

Za jednu z nejjednodušších bezeztrátových kompresí se považuje metoda *RLE* (zkratka anglického názvu Run-length encoding). Její princip spočívá v potlačení sekvence znaků, které dosáhnou potřebného opakování. Komprimace probíhá tak, že opakující se stejný znak je nahrazen indikátorem komprese, daným znakem a hodnotou určující počet jeho opakování. Indikátor je pomocná značka pro dekompresi, podle která se rozpozná zakódovaná sekvence. Výsledek komprese jsou data bez opakujících se sekvencí znaků.

Mezi nejznámější bezeztrátové komprese patří Golomb-Rice, Exp-Golomb a Huffman, které se využívají v multimédiích. Tyto komprimační metody jsou pojmenovány podle svých tvůrců.

3.1 Kódování Golomb-Rice

Golombovo kódování [11] je typ statistické metody komprese vynalezený profesorem Solomonem W. Golombem. Kódování je vhodné pro rozložení hodnot tak, že nízké hodnoty jsou více běžné než vysoké. Používá laditelný parametr b k rozdělení zakódování na 2 části. První část zastupuje hodnota q jako podíl (anglicky quotient) výsledku dělení parametrem b. Druhá část r je zbytek (anglicky remainder) po tomto dělení. Zakódování je pak ve výsledku složeno ze zapsané bitové posloupnosti jedniček o počtu q, od další části je odděleno vložením bitu 0 a vyjádřením bitové hodnoty r o délce b.

Kódování Golomb-Rice kopíruje Golombovo kódování pouze s tím rozdílem, že laditelný parametr b může nabývat jenom hodnot $\log_2 M$, kde M je mocninou čísla 2. Tento speciální případ Golombova kódování poprvé popsal Robert Rice. Tímto omezením parametru b se stane kódování nesmírně efektivní při použití v počítačích. Aritmetická operace dělení na získání q je nahrazena bitovým posunem a výpočet zbytku r se získá aplikováním bitové masky (ukázka viz tabulka 3.1).

index	q	r	výsledek
0	0	0	0 00
1	0	1	0 01
2	0	2	0 10
3	0	3	0 11
4	1	0	1 0 00
5	1	1	1 0 01
6	1	2	1 0 10
7	1	3	1 0 11

Tabulka 3.1: Ukázka kódování Golomb s $b = 2$

Při dekomprimaci je potřeba znát laditelný parametr b. Každý zakódovaný znak obsahuje úvodních q bitů s hodnotou 1 zakončených bitem 0 a b bitů na konci. Výsledek se tak získá umocněním q^b a přičtením posledních b bitů.

Golomb-Rice kódování je využíváno v některých audio kodecích (FLAC, ALAC od Apple).

3.2 Kódování Exp-Golomb

Kódování Exp-Golomb [10] vychází z principů Golombova kódování. Navzdory obdobnému jménu se vzdáleně podobá kódování Golomb. Proces kódování probíhá následovně (vyjádření viz 3.1 a ukázka kódování viz tabulka 3.2):

1. Vybrané celé kladné číslo se vyjádří v bitech a přičte se aritmeticky 1. Výsledek k se zaznamená.
2. Získá se počet bitů p zaznamenaného výsledku k a odečte se od něm 1. Výsledek se zapíše jako posloupnost bitů s hodnotou 0.
3. Na konec posloupnosti se vloží bit 1. K výslednému zápisu se připíše prvních p bitů z k s nejnižším významem.

$$\begin{aligned} num &\implies [M\,0][1][INFO] \\ M &= \log_2(num+1) \\ INFO &= num + 1 - 2^M \end{aligned} \tag{3.1}$$

```
0 ⇒ 1 ⇒ 1
1 ⇒ 10 ⇒ 010
2 ⇒ 11 ⇒ 011
3 ⇒ 100 ⇒ 00100
4 ⇒ 101 ⇒ 00101
5 ⇒ 110 ⇒ 00110
6 ⇒ 111 ⇒ 00111
7 ⇒ 1000 ⇒ 0001000
8 ⇒ 1001 ⇒ 0001001
```

Tabulka 3.2: Ukázka kódování Exp-Golomb

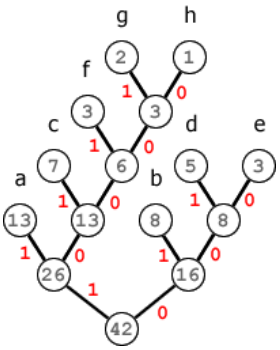
Dekomprimace se provádí podobně jako v případě Golombova kódování. Není nutno znát parametr b. Ten se získá sečtením prvních k bitů v úvodní posloupnosti nul oddělených od zbytku bitem `1`. Zbytek zakódovaného znaku tvoří k bitů, které se přičtou k číslu 2^k a snížením výsledné hodnoty 1 se získá zakódované číslo.

Využívá se ve standardu MPEG-4 část 10 (x264, Nero AVC...) a je také použito ve video kodeku Dirac.

3.3 Huffmanovo kódování

Roku 1952 byl publikován článek "Metoda pro konstrukci minimálně-redundantního kódu" (v originále "A Method for the Construction of Minimum-Redundancy Codes") od Davida A. Huffmana, který v té době působil jako student na MIT (Massachusetts Institute of Technology). V toto článku se zabývá tvorbou efektivního kódu, který nejvíce používá při bezeztrátovém kódování.

Huffmanovo kódování [12] [1] je podobně jako kódování Golombovo statistickou metodou, která kóduje data proměnlivou délkou kódu podle pravděpodobnosti výskytu. Během kódování se vytváří binární strom, kde hodnoty uzlů od kořene k listům stromu tvoří jednotlivá kódová slova. Jedná se tedy o tzv. prefixové kódy, což jsou kódy splňující podmínku, že žádné kódové slovo není počátkem (prefixem) jiného kódového slova. Algoritmus nejprve vezme dva listy/uzly s nejnižší pravděpodobností a umístí je jako listy binárního stromu (kódové slovo jednoho končí 1 a druhého 0). Listy pak ve statistice nahradí uzlem nad nimi a tomuto uzlu, který spravuje tyto dva listy, přiřadí součet pravděpodobností (těchto listů). Postup se opakuje až ve statistice zbyde jediný uzel. Názorná ukázka tvorby stromu je zobrazena na obrázku 3.2 a 3.1.



Obrázek 3.1: Konečná podoba Huffmanova binárního stromu

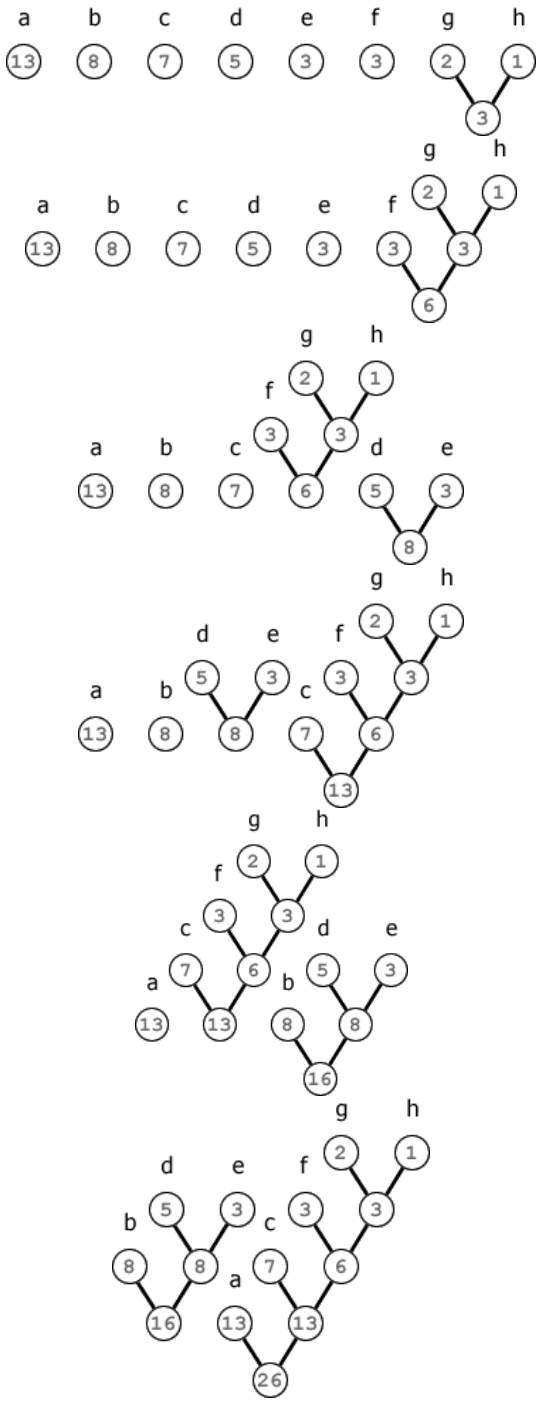
a	⇒	11
b	⇒	01
c	⇒	101
d	⇒	001
e	⇒	000
f	⇒	1001
g	⇒	10001
h	⇒	10000

Tabulka 3.3: Ukázka Huffmanova kódování vycházející ze stromu na obrázku 3.1

Dekomprimace probíhá následovně. Program přečte binární hodnotu prvního bitu. Je-li tato hodnota 1, vydá se v binárním stromu vlevo, je-li 0 pak vpravo - to se opakuje, dokud nedorazí až k listu stromu, kde nahradí kód daným slovem a pokračuje čtením bitů opět od kořene stromu.

Existují různé variace Huffmanova kódování. Jedna z nejpoužívanějších variací je adaptivní Huffmanovo kódování, které se liší od originálu tím, že je jedno-průchodové. Statistiku pravděpodobností si vytváří za běhu, a tím pádem neustále přebudovává svůj kódovací strom.

Huffmanovo kódování není patentováno a má díky své jednoduchosti a efektivitě široké využití v různých oblastech počítačové techniky. Používá se ve standardu JPEG stejně jako ve všech standardech MPEG. Používá se jak v populárním audio formátu MP3, tak i například v přenosech dat po síti. Pozměněná verze našla svoje uplatnění i ve faxovací technice. Huffmanovo kódování se stalo téměř běžným základem každé složitější komprese.



Obrázek 3.2: Tvorba Huffmanova binárního stromu

Kapitola 4

Ztrátová komprimace

Obecný přístup ztrátové komprese je jednoduchý. Po úvodním předzpracování se přeskupí nebo transformují data tak, aby bylo možno lehce oddělit důležité informace od nedůležitých. Méně důležité informace se pak potlačí mnohem více než důležité a nakonec se výsledek zkomprimuje některým z bezeztrátových kompresních algoritmů. Odstranění méně důležitých dat určuje míru zkreslení, které při vysokých hodnotách způsobuje komprimační artefakty. Využívá se především pro kompresi dat určených k smyslovému vnímání (obraz, zvuk, video).

Metody ztrátové na rozdíl od bezeztrátových prožívají v posledních několika letech bouřlivý rozvoj a budou mít velkou váhu i v letech následujících. Umožňují totiž digitalizovat zvuk, film, fotografie. Dnes si lze těžko představit, že by počítač obsahoval pouze textový editor a tiskárnu, nejvýše nějaký kompilátor některého z programovacích jazyků. Dnešním standardem se stává multimediální počítač, který umožňuje přehrávání videa, hudby a případně hraní počítačové hry. Počítače jsou používány stále více ve filmu nebo k úpravě fotografií. A právě toto je možné díky ztrátové kompresi dat.

Základem každé ztrátové komprese je transformace. Ta většinou vychází z nějaké matematické funkce, která transformuje data do domény (frekvenční, časové...). Při samotné transformaci se data neztrácejí. Ke ztrátě dat dochází až při procesu, který se označuje jako kvantizace. Kvantizace je postup, při kterém dojde k omezení hodnot a následně ztrátě dat. Kvantizace dělí koeficienty vypočtené transformací a následně je zaokrouhlí. Míra ztráty dat ovlivňuje velikost koeficientů kvantizace. Výsledné koeficienty po aplikaci kvantizace jsou načteny dle specifického sledu. Každá transformace má svůj specifický způsob načtení koeficientů, aby se dosáhlo co největší míry kompresibility. Načtená data jsou potom zakódována některou z bezeztrátových kompresí.

4.1 DCT

Na začátku 80. let se na scéně objevil standard JPEG (Joint Photographic Experts Group) pojmenovaný podle skupiny, kterou byl vytvořen. Byl to zároveň první krok pro využití

ztrátových kompresních metod v grafice, který přetrvává dodnes. Přinesl sebou diskrétní kosinovou transformaci (anglicky discrete cosine transform, zkráceně DCT [4] [8] [9]).

DCT vychází z diskrétní Fourierovy transformace, ale pracuje pouze s reálnou složkou. Je jednou z mnoha transformací příbuzných Fourierově transformaci. Existuje 8 standardních variant DCT, z nichž 4 jsou běžně používané. Pro kompresi obrazu se používá DCT typu II (častěji uváděná jako "DCT", viz rovnice 4.1) a k ní inverzní DCT typu III (častěji uváděná jako "iDCT", viz rovnice 4.2).

$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\Big(n+\frac{1}{2}\Big)k\right] \quad k = 0,\ldots,N-1 \tag{4.1}$$

$$X_k = \frac{1}{2}x_0 + \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}n\Big(k+\frac{1}{2}\Big)\right] \quad k = 0,\ldots,N-1 \tag{4.2}$$

Ve vzorci výpočtu DCT představuje výraz 4.3 vyjádření *bázové kosinové funkce*. Bázové funkce jsou ortogonální (skládají se z kolmých úhlů). Pro zadaný řád transformace N mají bázové funkce typický průběh. Při větší hodnotě N jsou vzorky bázových kosinových funkcí zobrazeny s vyšší hustotou. Bázové funkce mají hlavní podíl na transformaci hodnot do určité domény. V případě DCT dochází k přesunu složek do frekvenční domény. Na obrázku 4.1 jsou ukázány bázové funkce dvourozměrné DCT. Hodnoty funkcí jsou vyjádřeny stupněm šedi.

$$\cos\left[\frac{\pi}{N}\Big(n+\frac{1}{2}\Big)k\right] \tag{4.3}$$

Pro kompresi obrazu se používá dvourozměrná DCT. Výpočet dvourozměrné DCT typu II zobrazuje rovnice 4.4. Obraz je rozdělen do segmentů (bloků) o velikosti $N \times N$ (formát JPEG používá segment o velikosti 8×8) a nad každým segmentem je provedena 2D DCT. Nedokonalostí DCT je vznik blokových artefaktů, které při vysokém stupni komprese viditelně ruší obraz. To je dáno právě segmentací obrazu. Ve video kompresi se například používá zvláštní filtr na odstranění viditelných přechodů mezi segmenty.

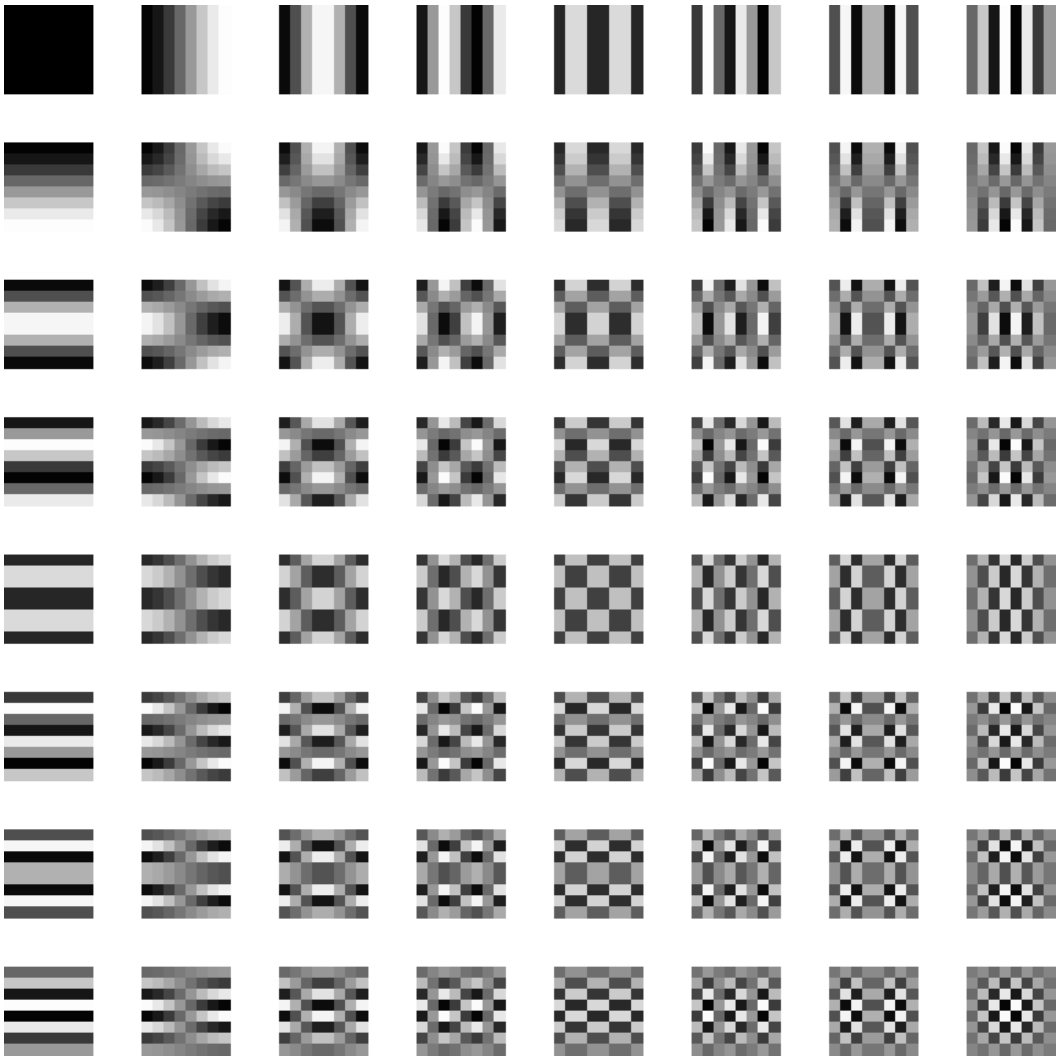
$$X_{k_1,k_2} = \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x_{n_1,n_2} \cos\left[\frac{\pi}{N}\Big(n_1+\frac{1}{2}\Big)k_1\right]\cos\left[\frac{\pi}{N}\Big(n_2+\frac{1}{2}\Big)k_2\right] \tag{4.4}$$

DCT převede hodnoty segmentu do frekvenční domény. Každý vypočtený koeficient DCT tak spadá do určité oblasti, která zastupuje jednotlivé obrazové struktury (viz obrázek 4.2). V levém horním rohu se nachází koeficient DC, který označuje stejnoměrnou složku. Na obrázku 4.1 je vidět, že hodnota DC má konstantní plochu. Ostatní složky bloku se označují jako střídavé (AC[1]). Přechodem z levého horního rohu na protější roh dochází

[1] Jména označení koeficientů pochází z historického používaní DCT pro analýzu elektrických obvodů se stejnosměrným proudem (direct current) a střídavým proudem (alternating current).

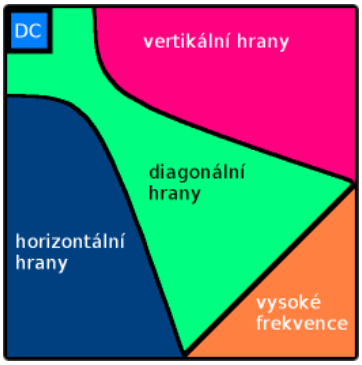
k narůstaní frekvence vzorků a ve výsledku vede ke zvýšení detailů v obrazovém segmentu. Tohoto jevu se využívá při kvantizaci.

Za nejdůležitější koeficient se považuje DC. Při odstranění AC koeficientů by tak vznikla po zpětné transformaci konstantní plocha. Důležitost koeficientů tedy plyne z toho, jak daleko se nacházejí od DC. Kvantizační tabulky jsou vytvořeny tak, aby obsahovaly nízké hodnoty u stejnosměrné složky i koeficientů amplitud kosinusových průběhů nízkých frekvencí a naopak vysoké hodnoty u koeficientů amplitud vysokých frekvencí. Výsledkem kvantizace bude nový blok hodnot, který většinou obsahuje u vysokých frekvencí nulové hodnoty.



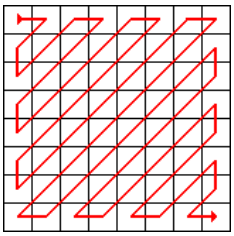
Obrázek 4.1: Bázové funkce

Po kvantizaci DCT koeficientů vznikne blok (matice) o rozměrech $N \times N$. Tento blok je potřeba převést do lineární podoby. V rámci standardu JPEG byla navrženo pořadí čtení hodnot z bloku. Postup linearizace zohledňuje důležitost jednotlivých koeficientů v jednom segmentu (koeficienty s větším významem jsou načítány jako první). Tento postup čtení je



Obrázek 4.2: Zastoupení DCT koeficientů v bloku

znázorněn na obrázku 4.3 a označuje se jako cik-cak sekvence (anglicky zig-zag). Výsledná sekvence se zbaví koncových nul a zbývající hodnoty se zkomprimují některou z bezeztrátových komprimací.



Obrázek 4.3: Cik-cak sekvence

DCT je často používána pro zpracování signálu nesoucí informaci smyslového vnímání. Je použita ve formátu JPEG, video kodecích MJPEG, standardech MPEG (DivX, XivD, x264) a dalších. Modifikovaná verze DCT typu IV se stala základem komprese audio, která zaznamenala veliký rozvoj příchodem známého formátu MP3.

Kapitola 5

Implementace

Video je sekvence obrázků zobrazovaných rychle za sebou tak, že v oku vzniká iluze pohybu. Frekvence zobrazování je v Evropě 25 snímků za sekundu. Pomocí prokládaného zobrazování lichých a sudých řádků je docíleno frekvence 50Hz (norma PAL). Jen pro doplnění se v americkém systému NTSC promítá na obrazovky frekvencí přibližně 60Hz, a to s 30 lichými a 30 sudými půlsnímky.

Při rozlišení videa 640×480 pixelů se získá $640 \times 480 = 307\,200$ obrazových bodů. Každý bod nese barevnou složku RGB o velikosti 24 bitů ($640 \times 480 \times (24 \div 8) = 921\,600$). Vynásobením počtem zobrazovaných snímků za sekundu datový tok vzroste na $640 \times 480 \times (24 \div 8) \times 25 = 23\,040\,000$ bajtů/s $\doteq 22$ MB/s. Při zachování toku dat by se 20 min video uložilo na 25 GB. Obsahem této práce je snížení datového toku za použití komprimačních metod a poznatků lidského oka.

Pro realizaci implementace videokodeku bylo použito vývojové prostředí *Microsoft Visual Studio 2005*. Programovací jazyk *C++* byl zvolen s ohledem na možnosti objektově orientovaného programování a vysoký výkon, který je při zpracovávání většího objemu dat zapotřebí.

První kapitola je věnovaná obecnému popisu aplikace. Zabývá se objektovým návrhem a popisem komunikace mezi objekty během komprimace.

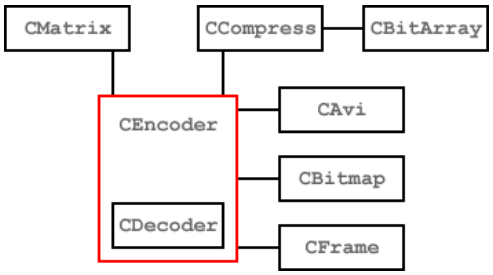
V následujících kapitolách budou probrány jednotlivé objekty, které se postupně podílejí na kompresi videa. Jsou zde popsány knihovny SDK (software development kit) jako je *Video for Windows* pro práci s formátem AVI a *DirectShow* objekty použité při tvorbě dekódovacího filtru.

5.1 Návrh aplikace

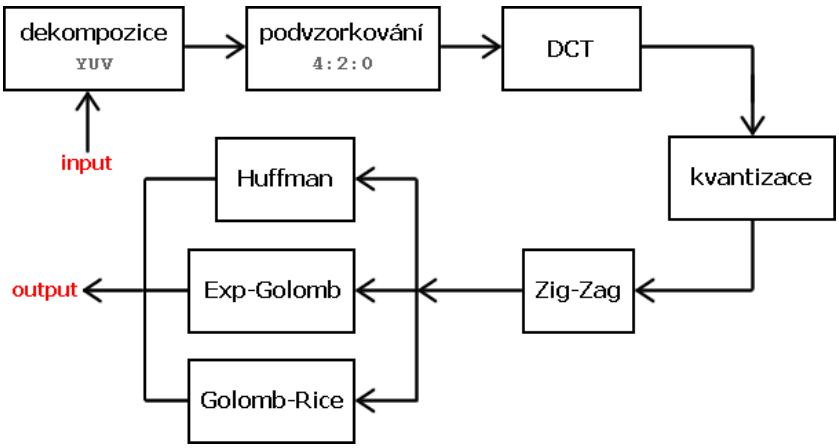
Při objektovém návrhu byl kladen důraz na obecnost řešení. Aplikace byla navržena tak, aby se dala jednoduše rozšířit.

Komunikace mezi objekty dle obrázku 5.1 probíhá následovně. O práci se sekvencí videa se stará objekt `CAvi`. Každý získaný snímek je uložen jako bitmapa. Správné uchování

bitmapy zajišťuje objekt `CBitmap` komunikující s jádrem aplikace `CEncoder` a `CDecoder`. `CEncoder` dědí část atributů a metod z objektu `CDecoder`, který provádí zpětnou komprimaci videa. V jádru aplikace se převede bitmapa na objekt `CFrame`. `CFrame` je speciálním úložištěm každého snímku videa. Pomocí objektu `CFrame` jsou získávány data pro `CMatrix` provádějící ztrátovou komprimaci. Výstupní data z `CMatrix` převezme objekt `CCompress` a společně s objektem `CBitArray` zkomprimují data zvolenou bezeztrátovou komprimací. Odtud data putují zpět do objektu `CAvi`, který je zapíše do výsledného videa. Jednotlivé body komprese ukazuje blokové schéma na obrázku 5.2.



Obrázek 5.1: Objektový návrh aplikace



Obrázek 5.2: Blokové schéma komprese

5.2 CAvi

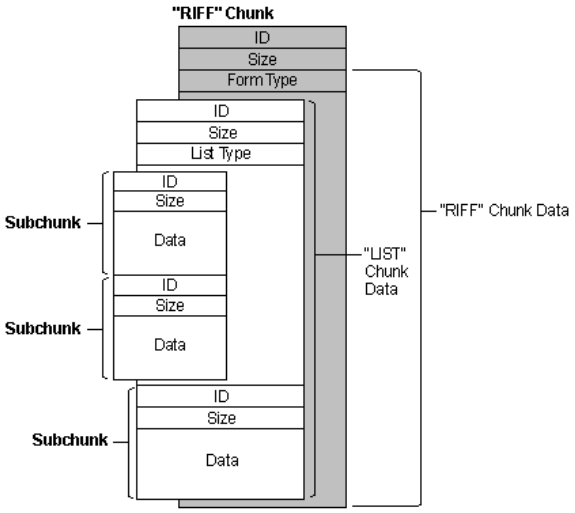
Jako zdroj videa je použit formát AVI (zkratka anglického názvu Audio Video Interleave). AVI je v současnosti jedním z nejrozšířenějších a nejpopulárnějších formátů videa. AVI formát ve své podstatě formátem RIFF (Resource Interchange File Format), který je souborový formát firmy Microsoft pro ukládání multimediálních zvukových a obrazových předloh. Soubor formátu RIFF je předznamenán signaturou “RIFF” na začátku souboru.

RIFF se skládá z datových struktur zvaných shluky (anglicky chunk); každý shluk má svoji čtyřznakovou signaturu (ID) definovanou v hlavičce shluku. Vnitřek shluku tvoří data a informace o jejich velikosti. Shluk může obsahovat tzv. podshluk (subchunk). Tímto způsobem je v AVI umožněna existence několika různých multimediálních dat. AVI formát umožňuje uložení videa, audia, textu a formátu midi.

První shluk AVI je pojmenován jako "hdrl". Obsahuje hlavičku AVI a další podshluky, které nesou informace o každém proudu dat (stream) uložených ve formátu. Každý stream obsahuje svou vlastní hlavičku s informacemi o datech, která nese. Pro jednoznačnou identifikaci dat se používá speciální značka FOURCC o 4 bajtech. Podle této značky se zjišťuje existence dekódovacího filtru v operačním systému a začlenění existujícího filtru do dekódovacího procesu. Více o tomto tématu pojednává kapitola 5.8 na straně 25.

Za prvním shlukem je uložen další, označený jako "movi", který tvoří skutečná data videostreamu. Za ním následuje "idx1" s informacemi o umístění pozic dat streamu tak, aby se mohlo efektivně přecházet mezi jednotlivými vzorky dat v souboru.

Práci s formátem AVI zajišťuje objekt `CAVI`, který objektově obaluje multimediální rozhraní *Video for Windows* dále jen VfW. Čte data snímků v dekomprimované podobě a vrací je jako bitmapu, se kterou se dále v programu pracuje. Vytváří také finální zakódované video.



Obrázek 5.3: Strutura formátu RIFF

5.2.1 Video for Windows

Video for Windows [5] je multimediální rozhraní vyvinuté v roce 1992 firmou Microsoft pro přehrávání digitálního videa. Společně s tím byl představen formát AVI. Rozhraní bylo poprvé vloženo do operačního systému Windows 3.1. VfW bylo vyvinuto v důsledek zvyšujícího se objemu dat v oblasti multimediální techniky. Pro programátory se stalo

základem manipulace s videem.

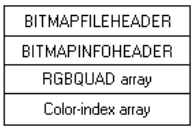
VfW je napsáno v jazyku C. Skládá se z množství funkcí, které zpracovávají video data z formátu AVI. Postupem času se nároky na možnosti práce s videem zvyšovaly, a tak v roce 1997 bylo VfW nahrazeno *DirectShow*. V současnosti se VfW už nevyvíjí, ale kvůli kompatibilitě je zachováno v operačním systému.

5.3 CBitmap

Bitmap je rastrovým obrázek popsaný pomocí jednotlivých barevných bodů (pixelů). Body jsou uspořádány do mřížky. Každý bod má určen svou přesnou polohu a barvu (RGB).

Struktura bitmapy je tvořena dle obrázku 5.4. Na začátku bitmap souboru je struktura BITMAPFILEHEADER, která obsahuje informace o typu souboru a jeho celkové velikosti. Za ním je uložena struktura BITMAPINFOHEADER s informacemi o šířce a výšce, počtu bitů určující barvu pixelů, použitá komprese a velikost dat následujících za touto strukturou. Ve složce RGBQUAD jsou uložena data s paletou barev použitých v bitmapě. Tato složka je použita pouze tehdy, pokud bitmapa obsahuje maximálně 256 barev, v ostatních případech má nulovou velikost. Až do konce souboru je bitmapa vyplněna daty o jednotlivých bodech bitmapy.

Veškeré operace nad bitmapou jsou zahrnuty v objektu `CBitmap`. Obsahuje metody pro ukládaní, načítaní a tvorbu bitmap.



Obrázek 5.4: Vnitřní uspořádání souboru s bitmapou

5.4 CFrame

Získaný snímek zdrojového videa je uložen jako bitmapa. Bitmapa je pro následnou komprimaci nevhodná a tak se převede na objekt `CFrame`. Objekt samotný funguje jako úložiště jednoho snímku videa včetně metod na jeho zpracování. Při převodu z bitmapy na `CFrame` dochází k dvěma zásadním změnám. Dekompozice, kdy je formát RGB převeden na barevný model YUV [14] a následné podvzorkování barevných složek [7].

Komprimovat jednotlivé složky RGB není vyhovující. Vhodnější je RGB převést na formát, kde je primárně vyjádřena složka jasu a složky barevného rozdílu. Tento převod vychází z možností lidského oka, které vnímá intenzivněji jas než ostatní složky. RGB model se proto převede na model YUV, který obsahuje složku Y reprezentující jas a složky U a

V jako barevný rozdíl. Vypočet YUV popisuje rovnice 5.1 a rovnice 5.2 vyjadřuje zpětný převod na formát RGB.

$$\begin{pmatrix} Y \\ U \\ V \end{pmatrix} = \begin{pmatrix} 0.299 & 0.578 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{5.1}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1.137 \\ 1 & -0.397 & -0.580 \\ 1 & 2.034 & 0 \end{pmatrix} * \begin{pmatrix} Y \\ U \\ V \end{pmatrix} \tag{5.2}$$

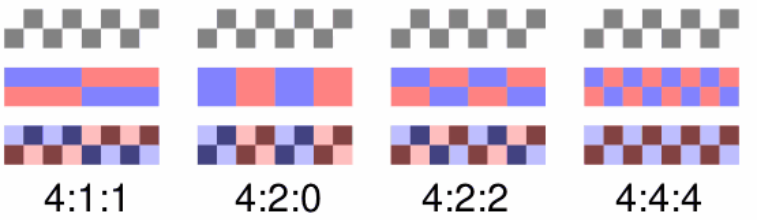
Vzhledem k tomu, že dekompozice se provádí u každého snímku, není příliš efektivní použití aritmeticky s plovoucí řadovou čárkou při výpočtu složek. Barevný model YUV byl navržen tak, že umožňuje přepsání výpočtu na základní aritmetické operace s celými čísly (viz rovnice 5.3 a 5.4).

$$\begin{aligned} Y &= ((66 * R + 129 * G + 25 * B + 128) \gg 8) + 16 \\ U &= ((-38 * R - 74 * G + 112 * B + 128) \gg 8) + 128 \\ V &= ((112 * R - 94 * G - 18 * B + 128) \gg 8) + 128 \\ \gg &\ \ldots \text{ bitový posun doprava} \end{aligned} \tag{5.3}$$

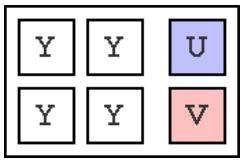
$$\begin{aligned} C &= 298 * (Y - 16) \\ D &= U - 128 \\ E &= V - 128 \\ R &= clip((C + 409 * E + 128) \gg 8) \\ G &= clip((C - 100 * D - 208 * E + 128) \gg 8) \\ B &= clip((C - 516 * D + 128) \gg 8) \\ clip() &\ \ldots \text{ oříznutí hodnoty do intervalu 0–255} \end{aligned} \tag{5.4}$$

Druhou změnou při převodu na objekt `CFrame` je podvzorkování barev. Jak již bylo řečeno, lidské oko vnímá více jas než barevné složky. Podvzorkováním snímku dojde ke ztrátě dat u barevných složek U a V, aniž by došlo k viditelné ztrátě kvality. Pro formáty YUV se vžilo tříčíselné označení, např. YUV 4:2:0. Udává vždy poměr mezi počtem barevné složky vůči jasové. V tomto případě je poměr 4:2:0 a barevná složka tedy obsahuje polovinu bodů vůči jasové - na dva jasové body připadá pouze jeden barevný (viz obrázek 5.5).

Při převodu z bitmapy je použito podvzorkování 4:2:0, při kterém dojde k 50 % úspoře dat. Podvzorkováním také vznikne nová základní jednotka snímku o velikosti 2×2 pixely, která se označuje jako **macroblok**. Obsahuje 4 jasové složky a po jedné složce barevného rozdílu. Ukázka obsahu jednoho macrobloku je zobrazena na obrázku 5.6.



Obrázek 5.5: Míry podvzorkování barevné složky



Obrázek 5.6: Macroblok

Konstruktoru objektu `CFrame` je předána bitmapa, která je převedena na barevný model YUV, podvzorkována a výsledek se ukládá do proměnné `frame_buffer`. Úložiště tvoří jednotlivé macrobloky (složky YYYYUV). Přístup k nim zajišťují metody `getPoint()`, `getPixel()` a metoda `setBlock()`, která nastavuje jeden určitý macroblok. Odlišnosti metod `getPoint()` a `getPixel()` jsou v typu vrácených dat. Metoda `getPoint()` vrací obrazový bod vyjádřený ve formátu YUV a metoda `getPixel()` vrací přímo převedený formát RGB. S těmito metodami potom pracuje objekt `Decoder` a z něj odvozený objekt `CEncoder`. Získaná data se předávají dalšímu objektu `CMatrix`.

5.5 CMatrix

Jelikož se komprimují data určená k smyslovému vnímání, našla své uplatnění ztrátová komprese DCT popsaná v kapitole 4.1 na straně 12. O správné provedení ztrátové komprese se stará objekt `CMatrix`. Objekt zejména spravuje matici o zadaných rozměrech, které jsou předány při inicializaci objektu.

Objekt `CMatrix` je nepostradatelnou součásti objektu `CDecoder` a všech objektů z něj odvozených. V těchto objektech je vytvořen hned na počátku. Důležitou součástí je proměnná `matrix`, která obsahuje spravovanou matici. Metodami `set()` a `get()` je možné měnit a získávat hodnoty matice. Metodou `clear()` se všechny položky matice nastaví na hodnotu zadanou parametrem. Pokud není parametr uveden, nastaví se všude hodnota 0. Zároveň zahrnuje metody implementující všechny fáze DCT.

Implementace DCT vychází ze vzorců uvedených v kapitole 4.1. Vzorce výpočtu transformace jsou modifikovány pro zvýšení efektivity. Rovnice 5.5 vyjadřuje výpočet dopředné DCT na jednorozměrném poli. Zpětnou DCT převádějící jednorozměrné pole do původní

podoby znázorňuje rovnice 5.6.

$$X_k = \frac{\sqrt{2}}{\sqrt{N}} C(k) \sum_{n=0}^{N-1} x_n \cos\frac{\pi(2n+1)k}{2N} \tag{5.5}$$

$$x_n = \frac{\sqrt{2}}{\sqrt{N}} \sum_{k=0}^{N-1} C(k)\ X_k \cos\frac{\pi(2n+1)k}{2N} \tag{5.6}$$

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{pro } k = 0 \\ 1 & \text{pro } k \neq 0 \end{cases}$$

K docílení rovnice výpočtu DCT pro dvourozměrného pole se musí zanést parametr definující druhý rozměr. Toho se dosáhne vynásobením dvou si odpovídajících rovnic s odlišným parametrem. Vynásobením vzniknou rovnice 5.7 a 5.8.

$$X_{k_1,k_2} = \frac{2}{N} C(k_1)C(k_2) \sum_{n_1=0}^{N-1}\sum_{n_2=0}^{N-1} x_{n_1,n_2} \cos\frac{\pi(2n_1+1)k_1}{2N} \cos\frac{\pi(2n_2+1)k_2}{2N} \tag{5.7}$$

$$x_{n_1,n_2} = \frac{2}{N} \sum_{k_1=0}^{N-1}\sum_{k_2=0}^{N-1} C(k_1)C(k_2)\ X_{k_1,k_2} \cos\frac{\pi(2n_1+1)k_1}{2N} \cos\frac{\pi(2n_2+1)k_2}{2N} \tag{5.8}$$

$$C(k) = \begin{cases} \frac{1}{\sqrt{2}} & \text{pro } k = 0 \\ 1 & \text{pro } k \neq 0 \end{cases}$$

Z povahy výpočtu obou rovnic je zřejmé, že nejvíce výpočetního času zabere počítání bázových funkcí (viz výraz 5.9). Ke zvýšení výkonu se výpočet bázových funkcí neopakuje a první výsledky se uchovají do dvou samostatných matic dle vzorců 5.10 a 5.11 [6]. Ze vzorců vyplývá, že CTt je transponovaná matice CT (anglicky cosine table). Závěrečný výpočet dopředné a zpětné DCT bez opakovaného počítaní bázových funkcí odpovídá násobení matic (viz rovnice 5.12 a 5.13). Inicializaci obou matic s hodnotami bázových funkcí provádí metoda `initDCT()`.

$$\cos\frac{\pi(2n_1+1)k_1}{2N} \cos\frac{\pi(2n_2+1)k_2}{2N} \tag{5.9}$$

$$CT[i,j] = \begin{cases} \frac{1}{\sqrt{N}} & \text{pro } i = 0 \\ \sqrt{\frac{2}{N}} \cos\frac{\pi(2i+1)j}{2N} & \text{pro } i > 0 \end{cases} \tag{5.10}$$

$$CTt[i,j] = CT[j,i] \tag{5.11}$$

$$DCT = CT \cdot A \cdot CTt \tag{5.12}$$

$$A = CTt \cdot DCT \cdot CT \tag{5.13}$$

I přes tuto optimalizaci není výpočet DCT dostatečně rychlý, aby zvládl plynulé dekódování videa. Aplikace je proto doplněna o úsek kódu podle dokumentu [2], který pro bloky o rozměrech 8×8 počítá DCT pomocí celočíselné aritmetiky.

Hlavní metodou provádějící dopřednou DCT nad maticí podle rovnice 5.12 je `DCT()`. Kvantizaci matice vykoná metoda `quantization()` s parametrem určujícím kvantizační matici. Při kvantizaci se dělí hodnoty matice hodnotami kvantizační matice (viz rovnice 5.14). Po kvantizaci se pomocí metody `zigzag_scan()` převede matice do lineární podoby. Nově vytvořené pole se předá jako odkaz parametrem včetně jeho velikosti. Od tohoto místa je další zpracování předáno objektu `CCompress()`.

$$\begin{aligned} A[i,j] &= round\Big(\frac{A[i,j]}{Q[i,j]}\Big) \\ round() &\ldots \text{ zaokrouhlení} \end{aligned} \tag{5.14}$$

Objekt `CMatrix` obsahuje také inverzní metody, které se používají při dekódování. Metodou `zigzag_fill()` je naplněna matice z pole sekvencí cik-cak s počátkem na souřadnicích $[0, 0]$. Ostatní hodnoty jsou nastaveny na 0. Násobením hodnot matice kvantizační maticí předanou parametrem zajišťuje metoda `dequantization()` a inverzní DCT vykoná metoda `iDCT()`.

5.6 CCompress

Jednotka provádějící sekundární kompresy po DCT se nazývá `CCompress`. Objekt `CCompress` obsahuje vnitřní pole, které se postupně zaplňuje přicházejícími daty z objektu `CMatrix`. Po nahrání všech dat se vybere jedna z metod bezeztrátové komprese podle nastavení kodéru. Každá z metod volá metody `analyze()` a na závěr `compress()`. Metoda `analyze()` projde pole s daty a vytvoří statistiku četností výskytu. Tu převezme jedna z metod bezeztrátové komprese a podle ní vybuduje bitové kódy. Metoda `compress()` pak prochází pole a nahrazuje jednotlivé hodnoty za bitové kódy. Výsledek je uložen v poli bitů, které obsluhuje objekt `CBitArray`.

Mezi metody bezeztrátové komprese objektu `CCompress` patří `GolombRice()` a k ní opačná `deGolombRice()`. Vychází ze způsobu komprese uvedené v kapitole 3.1. Vhodná hodnota laditelného parametru b se určuje automaticky podle rozložení hodnot ve statistice. Metody `ExpGolomb()` a `deExpGolomb()` zakódují pole kódováním Exp-Golomb, které je popsáno v kapitole 3.2.

Poslední metodou komprese je `Huffman()` a zpětná `deHuffman()`. Před tvorbou bitových kódu buduje binární strom (metoda `buildHuffmanTree()`). Metoda používá zásobník a frontu. Zásobník je na počátku zaplněn listy stromu seřazených podle četnosti výskytu. Na začátku jsou ze zásobníku vybrány dva listy s nejnižší četností. Z nich se vytvoří nový uzel s četností součtu obou listů a uloží se do fronty. V dalším kroku se vyberou listy nebo uzly s nejnižší četností a celý proces tvorby nového uzlu se opakuje. Proces končí

jakmile je zásobník prázdný a ve frontě se nachází pouze jediný uzel, který tvoří kořen celého stromu. Během tvorby stromu se při každém vytvoření nového uzlu volá metoda `setBranchCodes()`, která projde podstrom nově vytvořeného uzlu a aktualizuje bitový kód každé hodnoty listu. Vzorová ukázka tvorby bitového stromu je znázorněna na obrázku 3.2 a finální podobu včetně bitových kódů zobrazuje obrázek 3.1.

Zakódovaná data jsou vrácena jako pole bajtů. Pole se předá objektu `CAvi`, který je zapíše na uvedené místo výsledného videa.

5.6.1 CBitArray

Objekt `CBitArray` spravuje bitové pole. Velikost bitového pole se mění za běhu podle potřeby. Metodou `setBit()` se nastaví bit na hodnotu 1. Metoda `resetBit()` vynuluje bitovou hodnotu a metoda `toggleBit()` překlopí bit. Získání hodnoty bitu se provádí metodou `getBit()`.

Pro možnost zapisovat celé bitové kódy je objekt vybaven metodami `setBits()` a `getBits()`, které nastaví, popřípadě získají bity o zadané délce. Metodami `setArray()` a `getArray()` se vytvoří bitové pole z pole bajtů nebo se bity získají jako pole bajtů.

5.7 CDecoder a CEncoder

Objekty `CDecoder` a `CEncoder` jsou řídicími jednotkami aplikace. Objekt `CDecoder` slouží pro dekódování videa. Je součástí dekódovacího filtru, který bude popsán v následující kapitole 5.8. Objekt `CEncoder` dědí mnoho vlastností a metod z objektu `CDecoder`. Slouží výhradně k zakódování zdrojového videa.

Oba objekty obsahují metody pro předávaní dat mezi objekty. V průběhu komprese videa se získávají přes objekt `CAvi` snímky a ukládají se jako bimapy, které spravuje objekt `CBitmap`. Bitmapa je přeměněna na objekt `CFrame` s formátem YUV 4:2:0. Dál se v kompresi pokračuje podle nastavených parametrů předané konstruktoru objektu `CEncoder`.

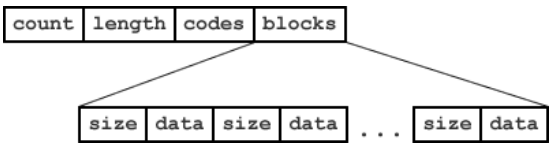
Parametry nastavení komprese definuje struktura `tOptions`. Určuje oblast videa, která se má komprimovat. Specifikuje velikost bloků, podle kterých je snímek rozdělen na segmenty. S každým segmentem (maticí) se provede DCT. Bezeztrátová komprese je vybrána z výčtového typu `tCompress`. Na kvalitu má největší vliv zvolený typ kvantizační matice a hodnota kvality, ze které se odvodí hodnoty kvantizační matice. Hodnoty kvantizační matice se počítají dle vzorce 5.15, kde q je hodnota kvality na stupnici od 0–100 a Q_{50} vystupuje jako střední hodnota kvantizační matice. Typy možných kvantizačních matic jsou FLAT, JPEG, MPEG, H263 a STD. FLAT matice obsahuje shodné hodnoty pro všechny koeficienty matice (hodnotu 16). JPEG matice vychází ze stejnojmenného formátu a obsahuje dvě samostatné matice. Jedna matice je použita na kvantizaci složky Y (jas) a druhá na zbylé složky U a V (barevný rozdíl). Matice MPEG se uplatňuje ve formátu MPEG-2 a matice H263 je základem formátu MPEG-4. Matice STD je standardní maticí, kde velikost koeficientů se zvětšuje přechodem z levého horního rohu na protější. Dalším parametrem

ovlivňující kvalitu je bit `zigzag_more`, který při cik-cak sekvenci aktivuje hlubší snímání. Dochází k zahazování nenulových hodnot, kterým předchází další sekvence nul.

$$\begin{aligned} Q_q &= \frac{Q_{50} \cdot (100 - q)}{50} \quad \text{pro } q = <50, 100) \\ Q_q &= \frac{50 \cdot Q_{50}}{q} \quad \text{pro } q = (0, 50) \end{aligned} \tag{5.15}$$

Po vytvoření objektu `CFrame` se z něj načítají bloky dat o zadaných rozměrech a předávají se objektu `CMatrix`. Blok dat je segment snímku, který tvoří vždy hodnoty z jedné složky formátu YUV. Vzhledem k podvzorkování barevných složek je počet složek Y dvakrát vyšší než složek UV. Načítají se proto nejdříve čtyři bloky složek Y, za kterými následují bloky se složkou U a V. S každým blokem dat se provede DCT pomocí objektu `CMatrix`. Vrácená data se zakódují vybranou bezeztrátovou komprimací (objekt `CCompress`) a uloží do výsledného videa.

Během procesu dekódování se provedou všechny kroky v opačném pořadí. Objekt `CDecoder` už nepotřebuje znát parametry komprese. Ty si zjistí sám z hlavičky videa společně s kvantizační maticí. Jsou mu postupně předávána data snímků tak, jak se vytvořily při kompresi (viz obrázek 5.7). Na začátku dat se nachází počet zakódovaných hodnot. Následuje délka kódovací tabulky a kódovací tabulka. Podle ní se dekódují bloky skládající se z velikosti a dat. Data nesou hodnoty, kterými je naplněna matice v rámci objektu `CMatrix`. Po aplikování inverzí DCT se získá blok dat pro objekt `CFrame`.

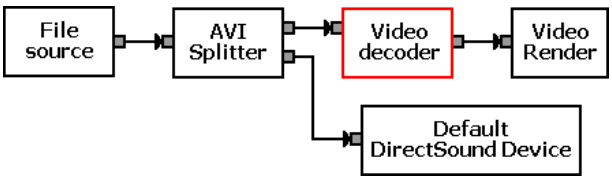


Obrázek 5.7: Strutura dat zakódovaného snímku

5.8 Dekodovací filtr

Integrace dekódovacího filtru do operačního systému Microsoft Windows je udělána pomocí architektury DirectShow [3]. DirectShow je balík COM (component object model) komponent, který v systémech Windows zpřístupňuje paletu multimediálních služeb. Jedná se o dynamický systém navzájem propojených komponent, které mají schopnost měnit své vazby podle aktuální potřeby. Umožňují využití různých hardwarových prostředků. Komponenty systému se označují jako filtry. Úkolem filtru je zpracování vstupních dat a předaní na výstup. Jednotlivé propojené filtry tvoří tzv. **filter graph**. O správné propojení filtrů se stará **filter graph manager**. Propojení filtrů funguje na bázi pinů. Každý filtr obsahuje jeden nebo více vstupních, výstupních pinů.

Pro účely dekódování videa se vytvoří následující graf viz obrázek 5.8. Na počátku grafu se čte soubor komponentou *File source* a binární obsah souboru předává dalšímu filtru. *AVI Splitter* rozděluje AVI souboru na dvě stopy. Většinou se jedná o zvuk a video. Zvuk je dále zpracováván filtrem *Default DirectSound Device*, který posílá zvuková data do zvukové karty. *Video decoder* dekóduje obrazovou stopu na formát dat, které převezme komponenta *Video Render* a pošle data přes grafickou kartu na obrazovku. Tvorba dekódovacího filtru spočívá ve vytvoření komponenty *Video decoder*.



Obrázek 5.8: Graf dekódování videa

DirectShow je balík objektových komponent. Dekódovací filtr je tedy objekt, který dědí vlastnosti z objektů `CPersistStream` a `CTransformFilter`. Objekt `CPersistStream` obsluhuje jednotlivé proudy dat, které jsou propojeny s filtrem. Druhý objekt `CTransformFilter` definuje rozhraní filtru. Obsahuje metodu `Transform()`, která převádí vstupní data na výstupní. Typ vstupních dat se kontroluje metodou `CheckInputType()` a typ dat pro výstupní pin určuje metoda `GetMediaType()`. Pro jednoznačné určení filtru se definuje GUID, které se skládá z několika hodnot, kde první 4 bajty tvoří FOURCC. FOURCC kód se při kompresi nastavuje ve formátu proudu videa tvořený strukturou BITMAPINFOHEADER. Ta obsahuje proměnnou `biCompression` nesoucí informaci o způsobu komprese. Každá takováto komprese je označena kódem FOURCC o 4 znacích (4 bajty). Tento videokodek používá označení “js-v”.

Pro vstupní pin stejně jako pro výstupní musí být zaručen hlavní typ video. Hlavní typ slouží k obecnému rozlišení dat (audio, video, text...). Podtyp zpřesňuje hlavní typ a pro vstupní pin je nastaven na GUID fitru včetně jeho FOURCC kódu. Podtyp výstupního pinu tvoří RGB24, který dokáže zpracovat filtr *Video Render*. Podle zvolených vstupních a výstupních typů (GUID) pak filter graph manager propojí komponenty do výsledného dekódovacího grafu.

Součástí DirectShow je i rozhraní `ISpecifyPropertyPages`, přes které se dá k filtru připojit okno s vlastnostmi. Na okno s vlastnostmi odkazuje metoda `GetPages()`, která musí být součástí filtru.

Proces dekódování probíhá v opačném směru než proces kódovaní. Charakter jednotlivých metod má opačný postup. Při kvantizaci se koeficienty násobí, pro DCT se volá inverzní transformace. Ve výsledku se získá video, které je podobné originálu. Podobnost závisí na kvalitě a typu kvantizační matice použité během komprese. Při nízkých hodnotách kvality dochází k degradaci obrazu a vzniku rušivých elementů. Jeden z nich je způsoben

segmentací obrazu, kdy dochází k viditelnému ohraničení bloků. Tomu je možno zamezit nasazením deblokovacího filtru, který projde okraje bloků a pozmění je tak, aby byl přechod plynulý.

Pro vlastní použití je nutné integrovat filtr do operačního systému Windows. To se provede příkazem `regsrv32 filtr.ax`, který zapíše potřebné informace do registru. Nyní je možné komprimované video přehrávat v softwarových přehrávačích videa.

Kapitola 6

Testování

Konečná aplikace se skládá z kodéru a dekódovacího filtru. Kodér tvoří program, který se ovládá z příkazové řádky. Vstupem a výstupem programu je formát AVI. Nastavení programu vysvětluje tabulka 6.1. Výchozím nastavením kodéru je kvalita `50`, kvantizační matice typu `flat`, bezeztrátová komprese `exp-golomb` a velikost segmentů snímku `8x8`.

`program -enc vstup výstup [nastavení]`	
`-frames start end`	vymezuje oblast videa
`-q [100-1]`	kvalita komprese
`-quant [flat,jpeg,mpeg,h263,std]`	typ kvantizační matice
`-compress [exp-golomb,golomb-rice,huffman]`	typ sekundární komprese
`-dct [4x4,8x8,16x16,32x32]`	velikost segmentů snímku
`-zigzag_more`	aktivuje hlubší snímání
`-help`	vytiskne nápovědu
`program -dec vstup výstup`	dekóduje video

Tabulka 6.1: Nastavení programu

Následující tabulky 6.2, 6.3, 6.4, 6.5 a 6.6 popisují vliv nastavení kodéru na výslednou velikost videa. Časy komprese uvedené v tabulce jsou měřeny na stroji s procesorem Intel Pentium M 2 GHz, 512 MB RAM a operačním systémem Windows XP. Testované video má 3 minuty a 14 sekund s velikosti 35,2 MB a je zkomprimováno kodekem XviD. Nejnižších velikostí se dosahuje s typem kvantizační matice STD a bezeztrátovou kompresí Exp-Golomb. Opakem je bezeztrátová komprese Golomb-Rice společně s kvantizační maticí typu FLAT, která má hodnoty matice konstantní a nezohledňuje důležitost koeficientů oproti kvantizační matici typu STD. Ukázky zkomprimovaných snímků jsou na obrázcích 6.1 a 6.2.

Naměřené časy ovlivňuje nejvíce výsledná velikost komprimovaného videa a rozměry bloků. V tabulce 6.6 je proveden test s hlubším snímáním při cik-cak sekvenci. Během hlubšího snímání dochází k zahazovaní nenulových koeficientů, kterým předchází sekvence nul. Tímto způsobem je snížena velikost na úkor kvality. Test komprese s různými rozměry bloků je zobrazen v tabulce 6.5. Pro bloky o rozměru 8×8 trvá komprese nejkratší dobu, protože DCT se počítá pomocí celočíselné aritmetiky. Pro ostatní rozměry platí, že čím větší rozměr, tím je výpočet DCT náročnější a trvá déle. Naopak u menších rozměrů nastává problém s kompresibilitou (viz obrázky 6.3).

`-q`	`-quant`	`-compress`	`-dct`	velikost	čas
80	flat	exp-golomb	8x8	175 238 kB	04:37,739
80	flat	golomb-rice	8x8	248 730 kB	05:00,221
80	flat	huffman	8x8	189 994 kB	04:37,509
80	jpeg	exp-golomb	8x8	129 329 kB	03:52,604
80	mpeg	exp-golomb	8x8	137 450 kB	04:20,063
80	h263	exp-golomb	8x8	133 624 kB	04:15,787
80	std	exp-golomb	8x8	110 730 kB	03:59,594
80	std	golomb-rice	8x8	166 999 kB	04:12,693
80	std	huffman	8x8	116 728 kB	04:06,143

Tabulka 6.2: Vliv nastavení na velikost při kvalitě 80

`-q`	`-quant`	`-compress`	`-dct`	velikost	čas
50	flat	exp-golomb	8x8	85 331 kB	03:20,037
50	flat	golomb-rice	8x8	103 710 kB	03:20,908
50	flat	huffman	8x8	87 218 kB	03:18,375
50	jpeg	exp-golomb	8x8	77 710 kB	03:15,070
50	mpeg	exp-golomb	8x8	77 907 kB	03:19,827
50	h263	exp-golomb	8x8	76 710 kB	03:18,555
50	std	exp-golomb	8x8	66 566 kB	03:10,073
50	std	golomb-rice	8x8	79 128 kB	03:12,556
50	std	huffman	8x8	66 297 kB	03:12,086

Tabulka 6.3: Vliv nastavení na velikost při kvalitě 50

-q	-quant	-compress	-dct	velikost	čas
20	flat	exp-golomb	8x8	45 125 kB	02:52,337
20	flat	golomb-rice	8x8	46 430 kB	02:55,101
20	flat	huffman	8x8	44 992 kB	02:51,326
20	jpeg	exp-golomb	8x8	44 163 kB	02:45,838
20	mpeg	exp-golomb	8x8	43 869 kB	02:53,159
20	h263	exp-golomb	8x8	43 655 kB	02:51,416
20	std	exp-golomb	8x8	39 443 kB	02:48,091
20	std	golomb-rice	8x8	41 811 kB	02:52,177
20	std	huffman	8x8	39 024 kB	02:48,011

Tabulka 6.4: Vliv nastavení na velikost při kvalitě 20

Obrázek 6.1: Originální snímek zakódovaný kodekem XviD

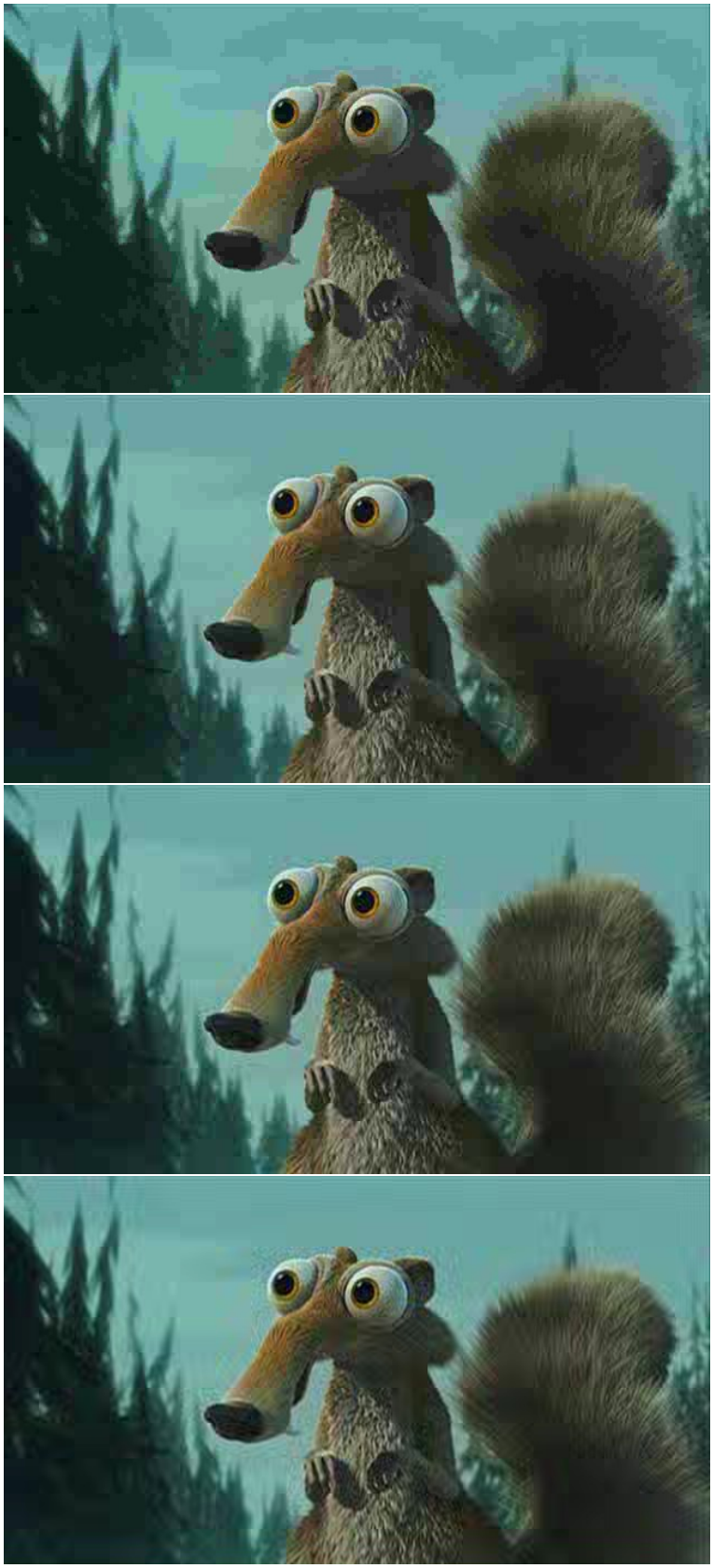
-q	-quant	-compress	-dct	velikost	čas
80	flat	exp-golomb	4x4	248 799 kB	07:42,525
80	flat	exp-golomb	8x8	175 238 kB	04:37,739
80	flat	exp-golomb	16x16	152 502 kB	09:33,975
80	flat	exp-golomb	32x32	169 882 kB	15:05,201
50	flat	exp-golomb	4x4	145 861 kB	06:20,957
50	flat	exp-golomb	8x8	85 331 kB	03:20,037
50	flat	exp-golomb	16x16	67 003 kB	08:18,576
50	flat	exp-golomb	32x32	76 290 kB	13:50,073
20	flat	exp-golomb	4x4	87 138 kB	05:38,757
20	flat	exp-golomb	8x8	45 125 kB	02:52,337
20	flat	exp-golomb	16x16	30 869 kB	07:57,947
20	flat	exp-golomb	32x32	37 455 kB	13:46,147

Tabulka 6.5: Vliv rozměrů bloků na velikost a čas

-q	-quant	-compress	-dct	-zigzag_more	velikost	čas
80	flat	exp-golomb	8x8	ne	175 238 kB	04:37,739
80	flat	exp-golomb	8x8	ano	156 527 kB	04:37,939
80	std	exp-golomb	8x8	ne	110 730 kB	03:59,594
80	std	exp-golomb	8x8	ano	95 055 kB	03:44,192
50	flat	exp-golomb	8x8	ne	85 331 kB	03:20,037
50	flat	exp-golomb	8x8	ano	71 570 kB	03:29,651
50	std	exp-golomb	8x8	ne	66 566 kB	03:10,073
50	std	exp-golomb	8x8	ano	57 724 kB	03:00,189
20	flat	exp-golomb	8x8	ne	45 125 kB	02:52,337
20	flat	exp-golomb	8x8	ano	35 968 kB	02:48,542
20	std	exp-golomb	8x8	ne	39 443 kB	02:48,091
20	std	exp-golomb	8x8	ano	33 116 kB	02:42,824

Tabulka 6.6: Hlubší snímání při cik-cak sekvenci

Obrázek 6.2: Ukázky snímků při stupních komprese 20, 50 a 80



Obrázek 6.3: Ukázky snímků s kvalitou 20 a rozměry bloků 4×4, 8×8, 16×16 a 32×32

Kapitola 7

Závěr

Tato bakalářská práce se zabývá kompresí videa a implementací vlastní kódovací aplikace. Úvodem je probrána historie vývoje komprese dat určených k smyslovému vnímaní. Je popsán formát JPEG a jeho nástupce JPEG2000. Z video formátů byl zmíněn standard MPEG. V teoretické části (kapitoly 3 a 4) se podrobně rozebírají různé metody bezeztrátové a ztrátové komprese.

Kapitola 5 vysvětluje návrh a způsob implementace jednotlivých objektů aplikace. Popisuje tvorbu a integraci dekódovacího filtru do operačního systému Windows. Výsledná aplikace obsahuje program kodér ovládaný z příkazové řádky a dekódovací filtr, který je součástí DirectShow grafu.

Testování výsledné aplikace je obsaženo v kapitole 6. Z provedených testů plyne, jak možnosti nastavení programu ovlivňují velikost a čas komprese videa. Porovnáním použitých bezeztrátových metod je zjištěno, že metodou Exp-Golomb se v tomto případě dosáhne lepších výsledků než s kompresí Huffman. To je dáno rozložením hodnot získaných po DCT. Při nastavení odlišných rozměrů bloků dochází ke snižování velikosti na úkor času a kvality. Čím větší rozměr transformovaného bloku, tím je výpočet DCT náročnější a při nízkém stupni komprese jsou více viditelné kompresní artefakty. Naopak při nižších rozměrech je výpočet DCT rychlejší, ale kompresibilita je velice nízká. Jako nejvhodnější se jeví rozměr 8×8, pro který existuje efektivní algoritmus výpočtu DCT. Velikost také ovlivňuje typ kvantizační matice. Největší velikosti se dosahuje s kvantizační maticí typu FLAT a nejmenší s kvantizační maticí typu STD, která má rozloženy velikosti hodnot podle důležitosti koeficientů.

Závěrem bych se chtěl zmínit o dalších možnostech rozšíření aplikace. Komprese probíhá na jednotlivých snímcích, které nejsou mezi sebou nijak provázány. Dalo by se v tomto směru aplikaci rozšířit o detekční jednotku pohybu, podle které by se kódovaly pouze ty části snímku, které se liší od předchozích. Další možností je nahrazení ztrátové komprese DCT modernější vlnkovou transformací (wavelet) použitou ve formátu JPEG2000.

Literatura

[1] Owen L. Astrachan. Huffman coding: A cs2 assignment. [online] http://www.cs.duke.edu/csed/poop/huff/info/, 2004.

[2] Yann Guidon. Optimised winograd dct for the f-cpu core 0. [online] http://f-cpu.seul.org/whygee/dct_fc0/dct_fc0.html, 2003.

[3] Aleš Keprt. Architektura directshow. [online] http://objekty.pef.czu.cz/2003/sbornik/Keprt2003.pdf, 2003.

[4] Syed Ali Khayam. The discrete cosine transform (dct): Theory and application. [online] http://www.egr.msu.edu/waves/people/Ali_files/DCT_TR802.pdf, 2003.

[5] MSDN. Video for windows. [online] http://msdn.microsoft.com/library/default.asp?url=/library/en-us/multimed/htm/_win32_video_for_windows.asp.

[6] Libor Tinka. Jak funguje jpeg - princip pod lupou. [online] http://www.pcsvet.cz/art/article.php?id=4919&search=DCT.

[7] Wikipedia. Chroma subsampling. [online] http://en.wikipedia.org/wiki/Chroma_Subsampling.

[8] Wikipedia. Discrete cosine transform. [online] http://en.wikipedia.org/wiki/Discrete_cosine_transform.

[9] Wikipedia. Diskrétní kosinová transformace. [online] http://cs.wikipedia.org/wiki/Diskrétní_kosinová_transformace.

[10] Wikipedia. Exponential-golomb coding. [online] http://en.wikipedia.org/wiki/Exp-Golomb.

[11] Wikipedia. Golomb coding. [online] http://en.wikipedia.org/wiki/Golomb_coding.

[12] Wikipedia. Huffman coding. [online] http://en.wikipedia.org/wiki/Huffman_coding.

[13] Wikipedia. Jpeg. [online] http://en.wikipedia.org/wiki/JPEG.

[14] Wikipedia. Yuv. [online] http://en.wikipedia.org/wiki/YUV.