VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

BEZEZTRÁTOVÁ KOMPRESE VIDEA

LOSSLESS VIDEO COMPRESSION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MICHAEL ADAM POLÁK**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. DAVID BAŘINA, Ph.D.**
SUPERVISOR

BRNO 2017

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2016/2017

Zadání bakalářské práce

Řešitel: **Polák Michael Adam**
Obor: Informační technologie
Téma: **Bezeztrátová komprese videa**
Lossless Video Compression
Kategorie: Zpracování obrazu

Pokyny:

1. Seznamte se s metodami bezeztrátové komprese obrazu a videa (predikce, kódování rozdílu, entropické kódování).
2. Navrhněte postup komprese videosekvencí s použitím vybraných metod.
3. Implementujte tento postup formou videokodeku pro různé multimediální frameworky (Video for Windows, DirectShow, FFmpeg).
4. Porovnejte dosaženou účinnost komprese se v současnosti používanými kodeky (Huffyuv, FFV1).
5. Diskutujte výsledky Vaší práce a výhody jednotlivých kompresních metod.

Literatura:

- R. J. van der Vleuten and S. Egner, "Lossless and fine-granularity scalable near-lossless color image compression", 25th Symp. Inform. Theory in the Benelux, (Kerkrade, The Netherlands), pp. 209-216, June 2-4, 2004.
- M. Weinberger, G. Seroussi, G. Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS", Hewlett-Packard Laboratories Technical Report No. HPL-98-193R1, November 1998, revised October 1999. IEEE Trans. Image Processing, Vol. 9, August 2000, pp.1309-1324.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání.

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 nepřepisovatelném paměťovém médiu (CD-R, DVD-R, apod.), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Bařina David, Ing., Ph.D.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2016
Datum odevzdání: 17. května 2017

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

L. S.

doc. Dr. Ing. Jan Černocký
vedoucí ústavu

Abstrakt

Táto práca sa zaoberá bezstratovou kompresiou videa. Sú v nej uvedené niektoré farebné modely a konverzie medzi nimi. Ďalej sa zaoberá prediktormi a ich implementáciou. Posledná teoretická časť obsahuje popis kontextovej kompresie a entropického kódovania s podrobnejším rozborom aritmetického kódovania. Nasleduje popis implementovaného kompresného postupu vo forme kodeku a detaily o vnútornej štruktúre. Tento kodek je potom otestovaný a porovnaný s aktuálne používanými kodekmi FFV1 a Huffyuv. Ukázalo sa, že zvýšením počtu kontextov sa zvýši kompresný pomer, avšak sa nepodarilo prekonať kodek FFV1, kodek Huffyuv bol vždy prekonaný.

Abstract

This thesis is describing methods of lossless video compression. It also contains the description and formulas for conversions of color models suitable for lossless compression. The last theoretical part describes context compression and entropy coding with emphasis on arithmetic coding. Following the arithmetic coder is the description and details about the implemented video codec. This codec is then tested and compared with the currently used FFV1 and Huffyuv codecs. The result of this thesis is that adding the number of contexts improves the compression ratio although it does not outperform the compression ratio of FFV1, the compression ratio of Huffuyv was always outperformed.

Klíčová slova

kompresia, video, bezstratová kompresia, farebné modely, prediktory, entropické kódovanie, aritmetické kódovanie, kontextová kompresia

Keywords

compression, video, lossless compression, color models, predictors, entropy coding, arithmetic coding, context compression

Citace

POLÁK, Michael Adam. *Bezeztrátová komprese videa.* Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Bařina David.

Bezeztrátová komprese videa

Prohlášení

Prehlasujem, že som túto bakalársku prácu vypracoval samostatne pod vedením pána Ing. Davida Bařinu, Ph.D. Uviedol som všetky literárne pramene a publikácie, z ktorých som čerpal.

.....
Michael Adam Polák
18. května 2017

Poděkování

Rád by som poďakoval vedúcemu práce pánu Ing. Davidovi Bařinovi, Ph.D. za čas a odborné konzultácie počas vypracovávania práce.

Obsah

Kapitola 1

Úvod

Obrázky a videá sa stali súčasťou nášho každodenného života odkedy väčšina ľudí vlastní telefón. Avšak telefóny majú jedno veľké obmedzenie a to veľkosť úložného priestoru. Preto je čím ďalej tým dôležitejšie hľadať spôsoby, akými sa dajú tieto multimédiá komprimovať. Každý obrázok či video obsahuje veľa redundancií, čo je prebytočná informácia, bez ktorej by sa stále dal obrázok či video zrekonštruovať do pôvodnej podoby. Napríklad 1 minúta videa v rozlíšení 4K má veľkosť 31,89GB bez použitia kompresie. Po použití kompresie má však veľkosť 375MB vo formáte MPEG.

Televízni operátori a inštitúcie zaoberajúce sa kultúrnym dedičstvom majú zásadne rozdielny pohľad na kvalitu uchovávaného digitálneho materiálu. Zatiaľ, čo televízni operátori sa spoliehajú na nedokonalosti videnia, kde si pozorovateľ nevšimne stratu kvality, organizácie ako napríklad UNESCO, na uchovanie dát pre ďalšie generácie, používa bezstratovú kompresiu. Ďalšou oblasťou, ktorá využíva tieto techniky je filmový priemysel, kde je dôležité uchovať, čo najlepšiu kvalitu na ďalšie úpravy alebo na zvyšovanie rozlíšenia, ktoré bude v budúcnosti bežne podporované. Vzhľadom na túto nemalú skupinu použitia je potrebné sa venovať bezstratovej kompresii, keďže cena úložného priestoru je vysoká.

Existujú dva typy kompresie: stratová a bezstratová. V bezstratovej kompresii sa žiadna informácia o obraze nestratí pri kompresii. Stratová kompresia funguje na princípe kompromisu kvality obrazu a veľkosti súboru, niektoré informácie sa strácajú a po dekompresii vzniká obraz s menšou kvalitou. Stratová kompresia sa používa výlučne v multimédiách najmä, ak nie je možné inak skomprimovať obraz do určitého priestoru a nie sú potrebné všetky jeho detaily. Kompresia má však svoju cenu – výpočtový výkon. Pri každom zobrazení je potrebné najprv video dekomprimovať, čo pri vysokých rozlíšeniach a trvaní videa môže trvať nekrátku dobu. Hlavným ukazovateľom účinnosti kompresie je kompresný pomer. Ten vyjadruje pomer veľkosti pôvodného videa oproti skomprimovanému.

Na dosiahnutie, čo najvyššej kompresie je potrebné najprv spraviť niekoľko úprav obrazu. Prvým krokom je konvertovať farebnú škálu na nejakú, ktorá je vhodnejšia na kompresiu. Nasleduje prediktor, ktorý odhadne hodnotu ďalšieho bodu v obraze, vypočíta rozdiel oproti skutočnej hodnote a nakoniec túto hodnotu zakóduje entropický kodér.

Cieľom tejto práce je vytvorenie bezstratového kodeku pre frameworky FFmpeg, Video for Windows a porovnať jeho účinnosť so súčasne používanými kodekmi Huffyuv a FFV1. Kapitola 2 popisuje techniky používané na bezstratovú kompresiu, ktoré sú neskôr použité pri implementácii. Kapitola 3 sa venuje usporiadaniu stavebných blokov a popisuje podrobnejšie implementáciu a použité nástroje. Preposledná kapitola 4 sa venuje dosiahnutým výsledkom v rámci tejto práce a porovnáva ich s aktuálne najpoužívanejšími bezstratovými kodekmi Huffyuv a FFV1.

Kapitola 2

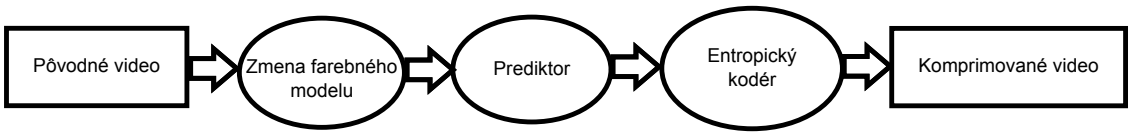
Komponenty kompresie videa

Na uvedenie čitateľa do problematiky je najprv nutné vysvetliť niektoré pojmy v bezstratovej kompresii. Nasledujúca kapitola popisuje jednotlivé kroky a komponenty na kompresiu videa a vysvetľuje ich funkciu a druhy implementácie.

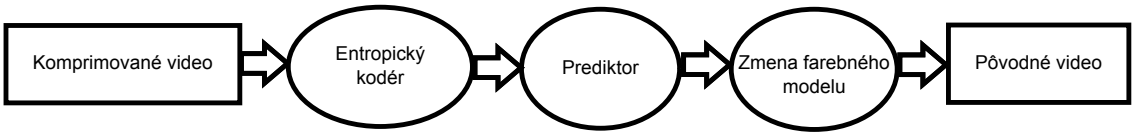
2.1 Bezstratová kompresia

Ako už bolo spomenuté v úvode, nie je vhodné pracovať so surovými dátami pretože zaberajú veľa priestoru. Z tohto dôvodu sa používa bezstratová kompresia, prostredníctvom ktorej sa dajú všetky informácie, ktoré boli prebytočné zrekonštruovať do pôvodnej podoby. V nasledujúcich kapitolách budú popísané metódy bezstratovej kompresie.

Bezstratová kompresia videa sa vykonáva v niekoľkých krokoch. Prvým krokom je konverzia farebného modelu RGB, ktorý nie je príliš vhodný na kompresiu pretože medzi jednotlivými zložkami je príliš blízky vzťah. Nasleduje prediktor, ktorý odhaduje hodnotu ďalšieho pixelu a nakoniec entropický kóder, ktorého výsledkom je komprimovaná snímka. Predikciu je možné robiť v rámci snímky, v prípade videa aj medzi jednotlivými snímkami. Medzi snímkami sa bude nachádzať značné množstvo redundancií, keďže na jednu sekundu pripadá približne 25 snímok, medzi ktorými je zvyčajne veľmi blízky vzťah. Avšak použitie rozdielových snímok je pre kompresný pomer pri použití bezstratovej kompresie nie vždy nevýhodné, bližšie je táto problematika popísaná v podkapitole 2.5. Kompresia a dekompresia videa prebieha podľa nasledujúcich schém 2.1 a 2.2.



Obrázok 2.1: Schéma kompresie videa.



Obrázok 2.2: Schéma dekompresie videa.

2.2 Farebné modely

Viditeľné svetlo [2] je elektromagnetické žiarenie v rozmedzí vlnových dĺžok od 380 nm do 780 nm. Farba sa z fyzikálneho hľadiska skladá z viacerých vlnových dĺžok jednotlivých farebných zložiek, ktorých zložním vznikne výsledná farba.

Farebný model v oblasti informačných technológii je matematický model, kde farba je reprezentovaná n-ticou čísel. Nasledujúce podkapitoly popisujú niektoré farebné modely používané v počítačovej grafike.

Základným farebným modelom, ktorý sa používa v zobrazovacej technike je inšpirovaný ľudským okom a nazýva sa RGB. Každý pixel obrazu je reprezentovaný hodnotou v rozsahu 0 – 255 pre každú z troch zložiek farby (červená, zelená, modrá). Tieto kombinácie nám ponúkajú možnosť reprezentovať viac ako 16 miliónov farieb. Nevýhodou takto uchovávaných hodnôt bodov je ich príliš blízky vzťah a z toho vyplývajúca redundancia. Taktiež pri zmene intenzity je potrebné daný bod celý prekresliť. Preto je lepšie transformovať tento farebný model na ekvivalentný, kde jedna zložka obsahuje informáciu o intenzite farby a zvyšné dve o farbe. Táto transformácia odstráni blízky vzťah jednotlivých zložiek. Pri spätnej transformácii však získame naspäť pôvodné hodnoty farby.

Model YUV je jeden z najbežnejšie používaných farebných modelov pre obraz a video [5]. Na rozdiel od modelu RGB obsahuje dve zložky s informáciou o farbe a tretiu o jase. Pre porovnanie model RGB obsahuje túto informáciu v rámci každej zo zložiek jednotlivých farieb. Tento model primárne vznikol na komunikáciu s analógovým televízorom, neskôr sa však začal používať aj na stratovú kompresiu. Prevod medzi RGB a YUV je možné vykonať pomocou vzťahov (2.1).

$$\begin{aligned} Y &= 0{,}299R + 0{,}587G + 0{,}114B \\ U &= 0{,}492(B - Y) \\ V &= 0{,}877(R - Y) \end{aligned} \tag{2.1}$$

Y je jas farby a U, V obsahujú informácie o farbe. Vzorec spätej konverzie na prehrávanie videa je vyjadrený vzťahmi (2.2).

$$\begin{aligned} R &= Y + 1{,}140V \\ G &= Y - 0{,}395U - 0{,}581V \\ B &= Y + 2{,}032U \end{aligned} \tag{2.2}$$

Model LYUV [10] je už vhodný farebný model na bezstratovú kompresiu, keďže všetky konverzie medzi RGB a LYUV dokonale reverzibilné. Na prevod medzi RGB a LYUV sa využíva vzorec, v ktorom sa výsledky zaokrúhľujú na najbližšie celé číslo. Nevýhodou použitia tohto modelu je zložitosť zmeny jasu jedného pixelu, tieto zmeny je potrebné urobiť najprv v modeli RGB a potom spätne previesť na model LYUV. Táto nevýhoda je však

pri kompresii zanedbateľná vzhľadom na to, že hodnoty nie je potrebné meniť. Konverzia farebného modelu je vyjadrená vzťahmi (2.3) a (2.4).

$$\begin{aligned} Y &= G + \frac{0{,}299}{0{,}587}R + \frac{0{,}114}{0{,}587}B \\ U &= B - 0{,}587Y \\ V &= R - 0{,}587Y \end{aligned} \tag{2.3}$$

$$\begin{aligned} B &= U + 0{,}587Y \\ R &= V + 0{,}587Y \\ G &= Y - \frac{0{,}299}{0{,}587}R + \frac{0{,}114}{0{,}587}B \end{aligned} \tag{2.4}$$

Tento model je štandardom pre stratovú aj bezstratovú kompresiu pre formát JPEG-LS [11] a nazýva sa LOCO-I. Jeho výhody i nevýhody sú rovnaké ako u modelu LYUV. Prevod z RGB je veľmi jednoduchý a je vyjadrený vzťahom (2.5).

$$\begin{aligned} C_1 &= R - G \\ C_2 &= G \\ C_3 &= B - G \end{aligned} \tag{2.5}$$

Spätná konverzia do RGB je vyjadrená vzťahom (2.6).

$$\begin{aligned} G &= C_2 \\ R &= C_1 + G \\ B &= C_3 + G \end{aligned} \tag{2.6}$$

Jedna z variant farebného modelu v štandarde JPEG 2000 [6] je bezstratová a nazýva sa JPEG 2000 RCT. Taktiež ako predošlé farebné modely dekoreluje vzťah medzi zložkami pixelu a oddelí ich na dve hodnoty obsahujúce informácie o farbe a tretiu ako informáciu o jase farby. Zložku obsahujúcu informáciu o jase je potrebné zaokrúhliť na najbližšie celé číslo smerom dole. Na transformáciu modelu z RGB sú potrebné vzťahy (2.7).

$$\begin{aligned} Y &= \left\lfloor \frac{1}{4}(R + 2G + B) \right\rfloor \\ C_b &= B - G \\ C_r &= R - G \end{aligned} \tag{2.7}$$

Spätná konverzia je vyjadrená vzťahom (2.8).

$$\begin{aligned} G &= Y - \left\lfloor \frac{1}{4}(C_b + C_r) \right\rfloor \\ B &= C_r - G \\ R &= C_b - G \end{aligned} \tag{2.8}$$

2.3 Prediktory

Prediktor slúži na odhad hodnoty aktuálne zakódovávaného pixelu použitím hodnôt z jeho okolia. Je to veľmi účinný nástroj na kompresiu videa. Umožňuje nám sústrediť hodnoty pixelov do okolia jednej hodnoty. Pokiaľ je prediktor dobre navrhnutý, budú odhadnuté hodnoty veľmi podobné skutočným alebo dokonca rovnaké. Všetky pixely sú pomocou prediktora zakódované ako rozdiel skutočnej hodnoty a predikovanej hodnoty pixelu. Je výhodnejšie zakódovávať hodnoty odchýlky prediktora namiesto skutočných hodnôt, keďže je pri videu vysoká pravdepodobnosť rovnakých rozdielov predikovaných hodnôt a skutočných hodnôt, ktoré môžeme zaznamenávať na menší počet bitov pomocou entropického kodéru. Rád prediktora určuje počet okolitých pixelov použitých na predpoklad hodnoty daného bodu obrazu. Pri dekompresii sa hodnoty odchýliek dané prediktorom spätne prepočítajú cez ten istý prediktor a sčítajú so zakódovanou hodnotou a výsledkom tejto operácie bude pôvodný bod obrazu. Táto technika sa tiež nazýva intra-frame kódovanie a slúži na zníženie priestorovej redundancie. Pri intra-frame kódovaní je jednoduchšie vyhľadávanie konkrétnych snímok vzhľadom na to, že nie je potrebné nájsť najbližšiu predošlú kľúčovú snímku a potom z nej dorátavať tú, ktorú je potrebné aktuálne zobraziť.

Na zakódovanie a odkódovanie hodnôt sa používajú vzťahy popísané v nasledujúcej tabuľke 2.1, kde e_x je chyba prediktora, P_x je predikovaná hodnota a X je skutočná hodnota pixelu. Pre okrajové hodnoty pixelov použitých pre predikciu sú hodnoty zložiek nulové alebo sú použité hodnoty z aktuálne predikovaného riadku alebo stĺpca. Ďalšou alternatívou pre okrajové hodnoty je použitie iného prediktora alebo použitie iného usporiadania pixelov na predikciu.

Zakódovanie	$e_x = X - P_x$
Dekódovanie	$x = e_x + P_x$

Tabulka 2.1: Vzťahy na zakódovanie a dekódovanie pri použití prediktora.

Lineárne prediktory používajú na predpoklad hodnôt nasledujúcich pixelov predošlé, už spracované a výsledkom predikovanej hodnoty je lineárna kombinácia hodnôt s cieľom znížiť priemernú druhú mocninu odchýliek prediktora. Na výpočet koeficientov pre lineárnu kombináciu existuje niekoľko metód, napríklad metóda Levinson-Durbin.

2.3.1 MED prediktor

Tento prediktor [11] je používaný algoritmom LOCO-I pri kompresii do bezstratového formátu JPEG-LS. Využíva hodnoty okolitých pixelov na predikciu nasledujúcich hodnôt podľa tabuľky 2.2. Tento prediktor slúži ako triviálny detektor horizontálnych a vertikálnych hrán. Keď prediktor detekuje vertikálne, horizontálne alebo žiadnu hranu vyberie hodnotu podľa poradia daného v nasledujúcom vzťahu (2.9). Táto implementácia prediktora umožňuje jednoducho pridať kontexty na zvýšenie kompresného pomeru.

		g	h
	c	b	i
d	a	x	

Tabulka 2.2: Konvencia značenia pixelov zvolených na predikciu.

Na výpočet hodnoty prediktora je použitý vzťah (2.9).

$$P_x = \begin{cases} \min(a,b), c \geq \max(a,b) \\ \max(a,b), c \leq \min(a,b) \\ a+b-c \end{cases} \tag{2.9}$$

Tretia hodnota v prediktore je gradientom, ktorá redukuje horizontálnu a vertikálnu redundanciu. Okrajové hodnoty, ktoré vyčnievajú za hranice snímky sú nahradené 0. Tento prediktor používa aj FFV1 kodek.

2.3.2 Paeth prediktor

Paeth prediktor vychádzajúci z Paethovho filtra je využívaný na predikciu hodnôt vo formáte PNG. Na predikciu aktuálne počítanej hodnoty využíva lineárnu kombináciu troch susedných pixelov. Funkcia sa snaží zistiť, v ktorom z troch smerov predikcie je gradient najmenší a túto hodnotu vráti. Funguje na podobnom princípe ako MED prediktor, pixely zvolené na predikciu sú popísané v tabuľke 2.2. Na výpočet predikovaných hodnôt sú použité vzťahy (2.11), (2.10).

$$\begin{aligned} p &= a+b-c \\ pA &= |p-a| \\ pB &= |p-b| \\ pC &= |p-c| \end{aligned} \tag{2.10}$$

$$P_x = \begin{cases} a, \min(pA,pB,pC) = pA \\ b, \min(pA,pB,pC) = pB \\ c \end{cases} \tag{2.11}$$

Alternatívne je možné používať škálovaný filter, kde každá zložka má váhu pre predikovanú hodnotu.

2.3.3 GAP prediktor

Lineárny prediktor GAP (gradient-adjusted predictor) [7][8] je používaný pri kompresnej metóde CALIC. Hlavnou výhodou prediktora GAP je vysoká schopnosť sa adaptovať, vďaka schopnosti detekovať slabé, pravidelné hrany, taktiež výrazné vertikálne a horizontálne hrany aj hladké oblasti obrazu. Jeho nevýhodou je však komplikovanosť a v prípadoch, kde obraz má veľké rozlíšenie trvajú výpočty dlhšie ako u predchádzajúcich prediktorov. Pixely na predikciu sú popísané v tabuľke 2.2.

Na výpočet hodnoty prediktora je použité vzťahy (2.12), (2.13).

$$\begin{aligned} d_h &= |a-d| + |b-c| + |b-i| \\ d_v &= |a-c| + |b-g| + |i-h| \end{aligned} \tag{2.12}$$

$$\bar{x} = \frac{a+b}{2} + \frac{i-c}{4} \tag{2.13}$$

Výsledné hodnoty prediktora sú dané podmienkami uvedenými vzťahom (2.14).

$$P_x = \begin{cases} a, d_v - d_h > 80 \\ \frac{\bar{x}+a}{2}, d_v - d_h > 32 \\ \frac{3\bar{x}+a}{4}, d_v - d_h > 8 \\ b, d_v - d_h < -80 \\ \frac{\bar{x}+b}{2}, d_v - d_h < -32 \\ \frac{3\bar{x}+a}{4}, d_v - d_h < -8 \end{cases} \tag{2.14}$$

2.4 Entropický kodér a kontextová kompresia

Najdôležitejšou časťou kompresie je entropický kodér. Entropický kodér zakódováva hodnoty pomocou kódu, ktorý je bez prefixu. Výhodou bezprefixového kódu je, že aj pri premenlivej dĺžke zakódovaných symbolov nie je potrebné vedieť dĺžku daného symbolu a je ho možné jednoznačne dekódovať. Dĺžka kódu je určená väčšinou podľa pravdepodobnosti výskytu jednotlivých znakov. Teda najčastejšie vyskytujúce sa znaky budú mať najkratšiu dĺžku. Podľa Shannonovej teorémy optimálna dĺžka kódu je určená vzťahom (2.15), kde b je počet zakódovaných symbolov a P je pravdepodobnosť symbolu na vstupe.

$$H_i = -\log_b P \tag{2.15}$$

Najznámejším entropickým kódovaním je Huffmanovo kódovanie, ktoré má dve varianty. Prvou variantou je statické kódovanie. Tento spôsob kódovania má od začiatku hodnoty pre symboly určené podľa priemerného výskytu. Druhou možnosťou kódovania je dynamické kódovanie, ktoré hodnoty symbolov určí podľa ich výskytov v danom prípade. Niektoré typy kódovania tiež berú do úvahy pravdepodobnosti nasledujúcich symbolov po aktuálne zakódovanom a priradia im iné hodnoty.

2.4.1 Kontextová kompresia

Kontexty [9] sú vzory okolitých pixelov, tieto pixely sa nachádzajú v rovnakej oblasti, ktorá už bola pred tým bola zakódovaná a tým pádom už adaptívny aritmetický kodér je inicializovaný počiatočnýni hodnotami zo snímky. V prípade, že existujú veľké odchýlky hodnoty prediktora a reálnej hodnoty pixelu v jednej oblasti je pravdepodobné, že aj nasledujúce hodnoty odchýlky budú taktiež veľké. Na druhej strane v prípade, že sa v jednej oblasti vyskytujú malé hodnoty odchýliek, je pravdepodobné, že aj odchýlka hodnoty aktuálne predikovaného pixelu bude malá v oblastiach snímky, kde sa intenzita mení postupne. Bolo odsledované, že veľké hodnoty odchýlok sa nachádzajú prevažne v oblastiach hrán obrazu. Tento fakt potom viedol ku záveru, že je vhodné voliť kontexty v závislosti od veľkosti odchýlky okolitých pixelov. Druhou možnosťou je kontexty voliť v závislosti od vetvy, ktorá je zvolená v prediktore vzhľadom na to, že tieto vetvy detekujú horizontálnu, vertikálnu hranu alebo hladkú oblasť.

Avšak používanie veľkého množstva kontextov má dve veľké nevýhody. Prvou nevýhodou je množstvo pamäti potrebnej na uchovanie veľkého množstva kontextov. Ak je potrebné rozlišovať n kontextov, kde každý z nich obsahuje pravdepodobnostný model je potrebné mať n krát viac pamäte vyhradenej na kontextové modely. Druhou nevýhodou je, že je potrebné inicializovať veľké množstvo kontextov, čo si vyžaduje istý čas. Tento čas sa bude výrazne predlžovať so zväčšujúcim sa množstvom kontextov. Toto veľké množstvo kontextov (väčšie ako 250) môže aj znížiť kompresný pomer vzhľadom na to, že pri 1000 hodnotách

sa môžu v jednom kontexte nachádzať iba 4 hodnoty. Avšak riešením tohoto problému je vhodná funkcia na určovanie kontextu, ktorý má byť použitý ako je to napríklad pri kodeku FFV1, ktorý používa kvantizačné tabuľky a má najlepšie výsledky týkajúce sa kompresných pomerov. Kontexty sú aktualizované priebežne počas zakódovávania hodnôt. Vzhľadom na to, že hodnoty sa budú sústrediť v okolí nulovej hodnoty je vhodné inicializovať všetky kontextové modely podľa Gaussovej krivky, ktorá sa bude najbližšie podobať distribúcii hodnôt po predikcii jednotlivých kanálov. Kontexty sú určované podľa zložky obsahujúcej jas, zvyšné zložky sú zakódované do toho istého kontextu pre svoj kanál.

2.4.2 Aritmetické kódovanie

Aritmetické kódovanie [3][13] dosahuje najlepší kompresný pomer, pretože zakódováva informácie najkompaktnejšie. Tento spôsob kódovania jednoznačne oddeľuje modely na reprezentáciu dát a ich zakódovanie. Na bezstratové zakódovanie je teda potrebné pracovať s informáciami, ktoré sú vopred známe kodéru aj dekodéru, napríklad platné rozsahy čísel, do ktorých sú hodnoty kódované. Jeho hlavnou výhodou oproti Huffmanovmu kódovaniu je zbavenie sa obmedzenia závislosti zakódovávaných hodnôt na mocnine 2.

Aritmetický kodér zakódováva symboly v intervale reálnych čísel od 0 do 1. Čím je počet možných symbolov väčší, tým menším intervalom čísel je možné reprezentovať sekvenciu symbolov. Každý nasledujúci symbol správy skracuje interval v závislosti od pravdepodobnosti výskytu symbolu vygenerovaných modelom na reprezentáciu sekvencií. Nasledujúca kapitola popisuje príklad kódovania s neobmedzenou presnosťou desatinných čísel a potom nasleduje kapitola popisujúca implementáciu v celých číslach.

2.4.3 Príklad aritmetického kódovania

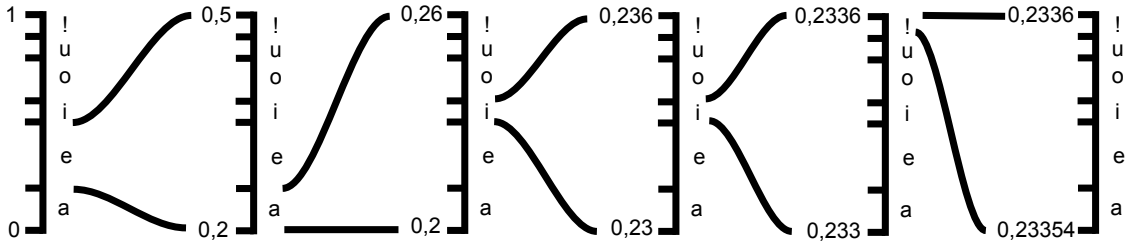
Pred začiatkom kódovania je rozsahom celý interval [0; 1). Po každom spracovanom symbole je tento interval zúžený o časť vyhradenú pre daný symbol. Príklad kódovania a dekódovania je predvedený na sade vstupných symbolov {a, e, i, o ,u, !}.

Symbol	Pravdepodobnosť	Interval
a	0,2	[0; 0,2)
e	0,3	[0,2; 0,5)
i	0,1	[0,5; 0,6)
o	0,2	[0,6; 0,8)
u	0,1	[0,8; 0,9)
!	0,1	[0,9; 1)

Tabulka 2.3: Príklad s fixným modelom pre samohlásky [13].

Zo začiatku kodér aj dekodér predpokladajú rozsah z intervalu [0; 1). Po prijatí prvého symbolu *e* kodér zúži interval na [0,2; 0,5), čo je rozsah vyhradený modelom pre symbol. Druhý príchodzí symbol *e* zase zúži tento interval o jednu-pätinu (hodnota vyplýva z tabuľky 2.3). Tento krok spôsobí, že interval má rozsah [0,2; 0,26). Táto hodnota bola získaná z pôvodnej 0,3, kde jedna pätina je 0,06. Pre nasledujúci symbol *i* je vyhradený rozsah podľa tabuľky 2.3 [0,5; 0,6), čo po aplikácií ďalšieho zúženia je výsledný rozsah [0,23; 0,236).

Pri dekódovaní sa aplikuje presne opačný postup ako pri kódovaní. Najprv je potrebné získať symboly a intervaly, ktoré im prislúchajú. Následne je vybraný z výsledného kódu



Obrázok 2.3: Príklad aritmetického kódovania [13].

symbol a určí sa interval, do ktorého patrí. Podľa toho vieme hodnotu symbolu. Ďalším krokom je spätné prepočítanie kódu použitím rovnakej techniky ako pri kódovaní.

2.4.4 Celočíselná varianta aritmetického kódovania

Vyššie použitá technika s použitím desatinných čísel sa v praxi nepoužíva z dôvodu, že predpokladá neobmedzenú presnosť čísla, ktorú v informačných technológiách nemáme k dispozícii. Preto sa v implementácii používajú kladné celé čísla o rozsahu v závislosti od bitovej šírky čísla (väčšinou 16-bitov). Rozsahy sú rozdelené obdobne ako pri implementácii s použitím desatinných čísel.

V podkapitole 2.4.3 je popísaný fixný pravdepodobnostný model, ktorý má hneď na začiatku dané pravdepodobnosti príchodov znakov a počas kódovania sa nemenia. Tento spôsob implementácie však predpokladá, že všetky zakódovávané údaje sa skladajú z približne rovnakých dát. Keďže každá snímka videa je iná, je vhodnejšie použiť iný pravdepodobnostný model. Druhou možnosťou je urobiť analýzu všetkých vyskytujúcich sa symbolov a ich početností, avšak tento prístup by bol príliš časovo náročný a vzhľadom na veľkosti snímok a počty potrebných operácií je používaný zriedkavo.

V prípade, že vopred poznáme veľmi presné štatistiky práve kódovaného symbolu je najvhodnejšie použiť dekrementujúci pravdepodobnostný model [3]. Váhy jednotlivých možných symbolov sú upravované dynamicky, teda sa zníži jeho váha pri každom jeho príchode. Týmto spôsobom je najefektívnejšie využitý priestor pre zakódovávanie symbolov, vzhľadom na to, že nie je ani jeden rozsah zbytočne malý alebo veľký.

Najčastejšie používaným pravdepodobnostným modelom je adaptívny vzhľadom na kompromis rýchlosti a dosiahnutých kompresných pomerov. Adaptívny model používa konštantne sa meniaci model, kde je frekvencia každého symbolu odhadom pravdepodobnosti jeho príchodu v každom bode exekúcie programu. Z toho vyplýva, že každý príchodzí symbol je zaznamenaný a nový vypočítaný rozsah je použitý v nasledujúcich výpočtoch.

2.4.5 Škálovanie

Pri použití už popísaných metód na kódovanie nastáva problém, keď minimum a maximum daného intervalu konvergujú k sebe až do bodu, kde sa obe hodnoty zhodujú. Existuje však jednouché riešenie - škálovanie intervalov.

Hneď ako sa hodnoty minima a maxima intervalu nachádzajú v rovnakej polovici celého intervalu je garantované, že sa hodnoty nebudú nachádzať mimo tohto intervalu, keďže nasledujúce znaky tento interval len zmenšia. Preto je možné informáciu o polovici intervalu uložiť a vynechať pri ďalších výpočtoch. V prípade, že najvýznamnejšie bity minima a

maxima intervalu sú 0, jedná sa o E1 škálovanie [1]. Na výstup je odoslaná 0 a rozsahy intervalov sú bitovo posunuté doľava. Pri E2 škálovaní sú oba MSB bity 1 a na výstup je odoslaná 1 a hodnoty rozsahu sú bitovo posunuté doľava. E1 a E2 škálovanie však nie je použiteľné v prípade, že interval konverguje k stredu celého intervalu. V tomto prípade sú MSB extrémov intervalu rozdielne. V tomto prípade sa používa E3 škálovanie. MSB bity sa v tomto prípade nezmenia, ale odstráni sa nasledujúci bit a zvyšok je bitovo posunutý doľava. LSB je nastavený pri minime intervalu na 0 a maxime na 1.

2.5 Intra-frame kódovanie pri bezstratovej kompresii

Pri intra-frame kompresii videozáznamu je spracovaná každá snímka samostatne a nie je závislá na iných snímkach. Týmto spôsobom pracujú bezstratové kodeky Huffyuv a FFV1. Bezstratový kodek Lagarith používa nulové snímky, ktoré indikujú zmenu oproti predchádzajúcej snímke, čiže špecifickú verziu inter-frame kódovania.

Použitím inter-frame kompresie videozáznamu môžeme dosiahnuť výrazne lepšie kompresné pomery, ale je výpočtovo náročnejšia a využíva sa najmä pri stratovej kompresii videa. Inter-frame kódovanie pri bezstratovej kompresii videa je výhodné len pre statické videozáznamy, ktoré obsahujú malé a/alebo pomalé zmeny. Pri dynamických zmenách scény pre bezstratovú kompresiu je výhodnejšie použiť intra-frame kódovanie, vzhľadom na to, že to, čo získame na inter-frame kompresii stratíme pri korekcii predikcie každého pixelu a zároveň výrazne zvyšujeme výpočtovú náročnosť kodeku, a teda znižujeme jeho rýchlosť.

2.6 Huffyuv kodek

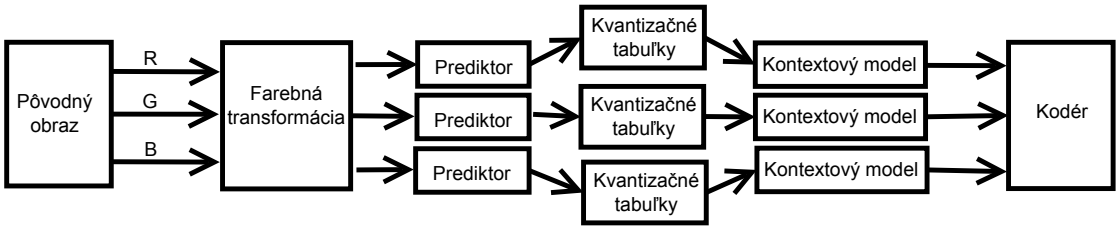
Huffyuv kodek vznikol ako náhrada neskomprimovaných hodnôt YCbCr, ktoré predtým sa používali na bezstratovú kompresiu. Najprv bol navrhnutý pre framework Video for Windows a neskôr pridaný aj do ďalších frameworkov. Jeho veľkou výhodou je jeho otvorenosť, keďže je aktuálne pod licenciou GPL. Je taktiež veľmi rýchly v porovnaní s kodekom FFV1, avšak nedosahuje také dobré kompresné pomery.

Na kompresiu používa techniku podobnú tej, ktorá sa používa v bezstratovej variante JPEG-LS. Podporované farebné modely sú: YCbCr, RGB a RGBA. Na predikciu používa prediktory v závislosti od použitého farebného modelu a v prípade RGB aj zároveň dekoreluje vzťah medzi zložkami. Následne použije Huffmanovo kódovanie na zákódovanie hodnôt. Hodnoty odchýliek kanálov snímky sú zakódované samostatne s použitím vlastného Huffmanovho stromu. Tieto stromy sú potom uložené do súboru spoločne s odchýlkami. Na zakódovanie stromov sa používa run-length kódovanie. Táto implementácia potom umožňuje dekomprimovať staršie súbory bez nutnosti explicitnej podopory starých Huffmanových stromov. Tento spôsob kódovania je tiež veľmi rýchly, umožňuje dekomprimovať až 38 megabajtov za jednu sekundu. Tieto stromy môžu byť aj vopred špecifikované aplikáciou na dosiahnutie lepšieho kompresného pomeru.

Tento kodek nepodporuje rozdielové snímky, teda každá snímka je zakódovaná zvlášť. Snímky síce zaberajú viac priestoru, ale vzhľadom na množstvo uchovávaných dát je potrebné minimalizovať množstvo výpočtov.

2.7 FFV1 kodek

FFV1 kodek [4] bol vyvinutý v roku 2003 pre framework FFmpeg. Je aktuálne najpoužívanejším kodekom na bezstratovú kompresiu vzhľadom na jeho otvorenosť (LGPL licencia) a dosiahnuté kompresné pomery, ktoré sú podobné pomerom kompresie JPEG 2000. Je primárne využívaný vzdelávacími a kultúrnymi inštitúciami na bezstratovú archiváciu videozáznamov. Výsledné skomprimované video má väčšinou tretinovú veľkosť oproti nekomprimovanému RGB. Ako väčšina bezstratových kodekov využíva len intra-frame kódovanie.



Obrázok 2.4: Blokový diagram FFV1 kodeku.

Bloková schéma 2.4 popisuje stavebné bloky kodeku. FFV1 používa farebné modely YCbCr alebo JPEG 2000 RCT na dekoreláciu hodnôt zložiek pixelu. Ako prediktor používa medián hodnôt, ktorý vypočíta predikovanú hodnotu z okolitých pixelov zobrazených v tabuľke 2.4. Bližšie je táto metóda popísaná v podkapitole 2.3.1. Tento kodek chýbajúce hodnoty mimo snímky v prípade potreby doplní nulami, to však značne zvýši odchýlku prediktora.

		T	
	tl	t	tr
L	l	X	

Tabulka 2.4: Kvantizované rozdiely vzoriek použité kontextom. X je aktuálne počítaná hodnota

FFV1 využíva veľký počet kontextov, ktoré slúžia na zásadné zvýšenie kompresného pomeru. Na určovanie kontextu sú použité kvantizačné tabuľky, v ktorých sa nachádzajú kvantizované rozdiely vzoriek snímky. Každá kvantizačná tabuľka obsahje presne 256 hodnôt a 8 LSB rozdielu vzorky je použitých ako index. Kontext je vypočítaný podľa vzťahu (2.16) a kvantizačná tabuľka je určená vzťahom (2.17).

$$\begin{aligned}\text{context} = Q_0[(l-tl)\&255] + Q_1[(tl-t)\&255] + \\ + Q_2[(t-tr)\&255] + Q_3[(L-l)\&255] + Q_4[(T-t)\&255]\end{aligned} \tag{2.16}$$

$$Q_i[a-b] = Table_i[(a-b)\&255] \tag{2.17}$$

Do výsledného súboru je uložená polovica kvantizačnej tabuľky. Nakoniec je aplikovaný entropický kodér, ktorý má implementovaných niekoľko variant: Huffmanovo kódovanie, run-length kódovanie a aritmetický kodér.

Na zvýšenie kompresného pomeru kodek prechádza jednotlivé snímky dvakrát. Prvým prechodom zanalyzuje vstupné dáta a zaznamená si štatistické výsledky a druhý prechod využije predom získané štatistiky na zvýšenie kompresného pomeru, avšak tento prístup má za cenu časovú zložitosť kodeku.

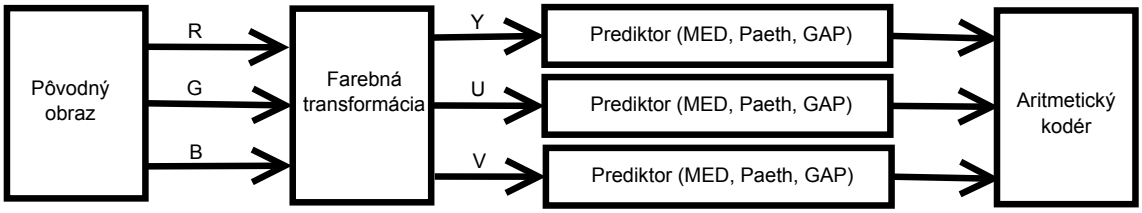
Každá snímka je kódovaná samostatne ako kľúčová. Na zvýšenie rýchlosti je algoritmus paralelizovaný a každá snímka rozdelená na niekoľko častí, ktoré sú kódované zvlášť, v špecifikácii sa tieto makrobloky nazývajú slice. Pridávanie ďalších vlákien pre zvyšovanie rýchlosti však znižuje kompresný pomer vzhľadom na to, že časti snímky sú samostatne kódované, a teda majú aj okraje navyše.

V tejto kapitole bol popísaný teoretický základ jednotlivých komponentov kodeku pre bezstratovú kompresiu videa. Boli popísané základné farebné modely, vybrané prediktory, princíp práce entropického kodéra a kodeky Huffyuv a FFV1, s ktorými budú výsledky tejto práce porovnávané.

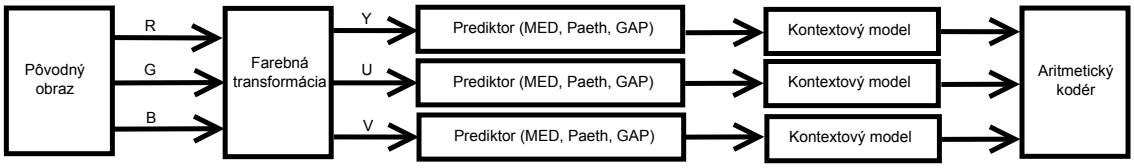
Kapitola 3

Návrh kompresného algoritmu

Cieľom tejto práce je implementácia bezstratového kompresného algoritmu pre framework FFmpeg (knižnica libavformat) a Video for Windows s cieľom dosiahnuť, čo najlepší kompresný pomer. Kodek musí obsahovať nasledovné stavebné bloky: farebná transformácia, prediktor, prípadne kontexty a nakoniec entropický kodér. V tomto prípade bol použitý aritmetický kodér. Vzťahy jednotlivých stavebných blokov sú zobrazené na obrázkoch 3.1 a 3.2. Pri dekompresii sú jednotlivé stavebné bloky rovnaké, ale aplikované v opačnom poradí. Pre implementovaný bezstratový kodek bol zvolený FourCC kód *MKP1*.



Obrázok 3.1: Schéma kompresného algoritmu pri použití prediktora.



Obrázok 3.2: Schéma kompresného algoritmu pri použití prediktora a kontextov.

Vzhľadom na to, že väčšina dnes používaných bezstratových kodekov využíva len intra-frame kódovanie bol v návrhu využitý tento prístup. Prvým krokom pri kompresii videa je dekorelácia farebného modelu (transformácia farebného modelu). V tomto prípade je na obrázkoch 3.1 a 3.2 farebný model RGB transformovaný na farebný model používaný kompresným algoritmom LOCO-I (2.2). Budú zvažované tri farebné modely LYUV, JPEG 2000 RCT a LOCO-I. Tieto farebné modely budú testované a do finálneho kodeku bude vybraný najvhodnejší. Nasleduje aplikácia prediktora pre každý farebný kanál zvlášť a výsledok je uložený do novej matice. Výber prediktorov pozostáva z trojice MED, Paeth a GAP bez kontextov a s kontextami. Opäť výkonnosť týchto prediktorov, čo sa týka kompresného pomeru

bude testovaná a do finálneho kodeku implementovaná najlepšia varianta. V prípade použitia kontextov sú hodnoty pre každý kanál zvlášť poslané do kontextového modelu zvoleným prediktorom, pre čo najlepší odhad pravdepodobnosti príchodu ďalšej hodnoty v danom kontexte. Ďalej je postupne zakódovávaný každý kanál zvlášť aritmetickým kodérom.

O načítanie a zápis obrazu sa starajú frameworky, ktoré poskytujú jednotlivé dekódované snímky vo farebnom formáte podľa určenia. V tejto implementácii je použitý farebný model RGB24, aby bolo možné jednoducho transformovať hodnoty na akýkoľvek iný model. Na zápis snímky je použitý buffer, do ktorého sú uložené už skomprimované dáta a framework ich správne uloží do kontajnera špecifikovaného pri kompresii. Jednotlivé snímky sú uložené nekomprimované v štruktúre, ktorá sa nazýva frame buffer. Frame buffer obsahuje hodnoty pixelov uložené v jednom riadku, kde medzi jednotlivými riadkami sa môžu nachádzať nevyužité bajty. Tieto medzery sa nazývajú stride. Štruktúra frame buffer má v sebe snímku uloženú zľava doprava, ale so spodným riadkom ako prvým. Túto vlastnosť je možné zmeniť parametrom tak, že snímka sa bude ukladať od prvého riadku.

Pred transformáciou farebného modelu je snímka konvertovaná z pôvodných 8-bitových RGB hodnôt pre jednotlivé kanály na 16-bitové (štruktúra *T_pixel*), pretože model LOCO-I využíva 9-bitové hodnoty pre reprezentáciu farby. Následne je farebný model prevedený z pôvodného RGB na LOCO-I použitím vzťahov z podkapitoly 2.2. Prevedené hodnoty sú zapísané do vyrovnávacej pamäte, s ktorou sa ďalej pracuje v nasledujúcich krokoch. Pre spätnú transformáciu farebného modelu je použitá funkcia, ktorej vstupom sú všetky kanály pixelu, ktorá zapíše výsledky späť do vyrovnávajúcej pamäte.

Na predikciu je možné použiť niekoľko funkcií v závislosti od použitého prediktora (podľa okolia ktoré použije na predikciu) a v závislosi od toho, či je použitá kontextová kompresia alebo nie. Jedným z parametrov každej funkcie je prediktor, ktorý má byť použitý. Implementované sú prediktory MED, Paeth a GAP, boli použité vo variante s použitím kontextov, aj bez nich. Pre okrajové hodnoty, kde prediktor vyžaduje hodnoty, ktoré sa nachádzajú mimo obrazu sú použité nuly inšpirované špecifikáciou pre kodek FFV1.

Posledným blokom navrhovaného kodeku je aritmetický kodér, ktorý bol vybraný na základe dosahovaných kompresných pomerov. Na aritmetické kódovanie a dekódovanie sú použité funkcie z knižnice pre aritmetický kodér, ako základ poslúžila knižnica [12]. Bola upravená pre podporu adaptívneho modelu, škálovania a ponechania skomprimovaných dát v pamäti namiesto zápisu priamo do súboru. Tento systém umožňuje existenciu viacerých kodérov a dekodérov a aj súčasné spracovanie viacerých snímok. Táto vlastnosť môže byť využitá pri kódovaní jednotlivých kanálov obrazu, kde pre každý kanál môže existovať samostatný kodér, a teda nie je potrebné celú snímku prechádzať trikrát po jednotlivých kanáloch. Takýto prístup je síce pamäťovo o niečo náročnejší, ale vzhľadom na počty operácii pre každú snímku je to zanedbateľné.

Kodek sa skladá z jadra, ktoré implementuje jednotlivé algoritmy podľa schém 3.1 a 3.2 a obálok, ktoré jadro prepoja s konkrétnym frameworkom.

3.1 Multimediálne frameworky

Framework je abstrakcia, kde software ponúka všeobecnú funkcionalitu, ktorá môže byť upravená pre potreby programátora a tým vytvárať programy pre špeciálne použitie. Niektoré frameworky pre prácu s videom sú prenosné (napríklad FFmpeg), iné sú špecifické pre platformu Windows (napríklad DirectShow alebo Video for Windows). V prípade frameworku FFmpeg na pridanie nového kodeku je potrebné kompilovať celú knižnicu, iné vyžadujú len zaregistrovanie novej časti. Frameworky zapúzdrujú kompresný algoritmus a

ukladajú obrazové a zvukové dáta do multimediálneho kontajnera. Kontajnery slúžia na ukladanie jednotlivých stôp (obraz, zvuk, titulky) do jedného súboru. Zabezpečujú synchronizáciu jednotlivých vzoriek zvuku a snímok videa s časovou osou. Vďaka tomu pri kompresii videa nie je potrebné sa zaoberať zvukovou stopou. Príkladom bežne používaných kontajnerov je MKV a AVI.

Na kompresiu jednotlivých snímok sú implementované funkcie, ktoré sú volané frameworkom podľa potreby. Tieto funkcie používajú parameter kontext, ktorý definuje základné informácie týkajúce sa výšky, šírky, farebného modelu a veľkosti zakódovaného paketu snímky atp. V prípade, že implementovaný kodek si potrebuje uchovávať informácie o okolitých snímkach, použitých prediktoroch, alebo kontextoch sa využíva takzvaný privátny kontext. Privátny kontext je dátová štruktúra, ktorá je prístupná len kodeku. Tento spôsob implementácie bol zvolený z dôvodu, že nie je potrebné, aby k týmto informáciám pristupovala aplikácia alebo iná časť rozhrania frameworku.

3.1.1 FFmpeg

FFmpeg je poprednou multimediálnou platformou. Dokáže dekódovať, kódovať, transkódovať, multiplexovať, demultiplexovať, streamovať, filtrovať a prehrávať prakticky akékoľvek video alebo zvukovú stopu. Podporuje rozsiahle portfólio formátov bez ohľadu na to, či boli vytvorené komisiou pre štandardy, korporáciami alebo komunitami. Je vysoko portovateľný, FFmpeg je implementovaný na mnohých platformách. Hlavným cieľom frameworku FFmpeg je poskytnúť čo najlepšie technické riešenie pre vývojárov aplikácií a používateľov. Je publikovaný pod licenciou GNU lesser general public license 2.1.

Na pridanie nového kodeku je potrebné upraviť knižnice libavcodec a libavformat. Ďalej je potrebné zaregistrovať nový kodek pomocou patchu. Nové kodeky sú šírené vo forme patchu. Patch obsahuje súbory ktoré sú nové, alebo rozdielne oproti pôvodným. Najdôležitejšia funkcia pre dekódovanie videa je *mkp_decode_init*, ktorá sa postará o inicializáciu aritmetického dekodéra a kontextu. Následne je volaná funkcia *mkp_decode_frame*, ktorá má na starosti dekódovanie jednej snímky videa. Na zakódovanie videa je použitá funkcia *mkp_encode_frame*, pre ktorú je potrebné vytvoriť rovnaký patch ako na dekódovanie videa. Príklad registrácie kodeku je uvedený nižšie.

```
AVCodec ff_mkp_decoder =
{
        .name           = "mkp",
        .long_name      = NULL_IF_CONFIG_SMALL("mkp␣/␣mkp"),
        .type           = AVMEDIA_TYPE_VIDEO,
        .id             = CODEC_ID_MKP,
        .priv_data_size = sizeof(MKPContext),
        .init           = mkp_decode_init,
        .close          = mkp_decode_close,
        .decode         = mkp_decode_frame,
        .capabilities   = CODEC_CAP_DR1,
};
```

Po aplikácii patchu je potrebné celú knižnicu znovu skompilovať a uistiť sa, že všetky komponenty boli správne nanovo preložené. Na zvolenie kodeku je použitý prepínač *-vcodec* s hodnotou *mkp*.

3.1.2 Video for Windows

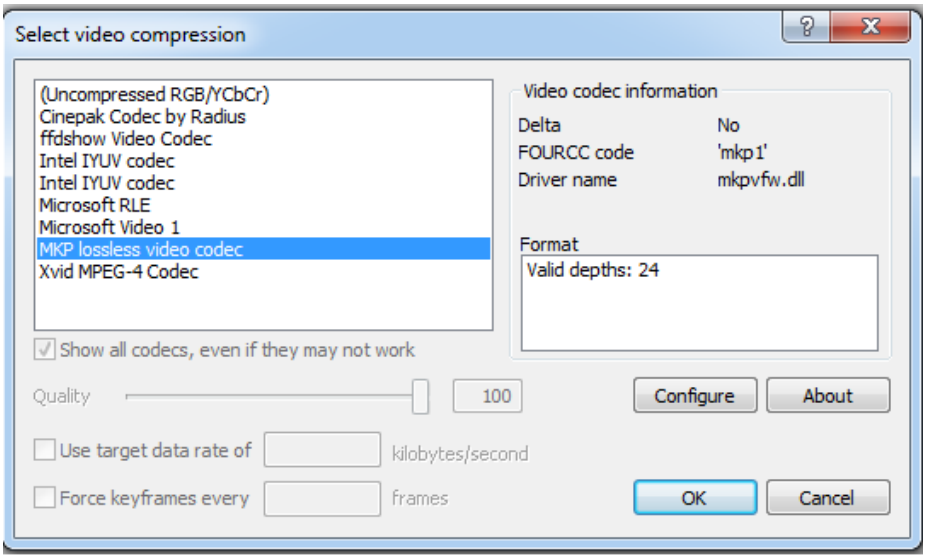
Framework Video for Windows od firmy Microsoft vznikol v roku 1991 ako reakcia na QuickTime od firmy Apple a bol uvedený v 16-bitovej verzii operačného systému Windows. Prvá verzia mala mnoho obmedzení, napríklad maximálne rozlíšenie 320 na 240 pixelov a maximálnu frekvenciou 30 snímok za sekundu. Video for Windows bolo v roku 1996 nahradené frameworkom ActiveMovie, neskôr DirectShow. Video for Windows ako prvý framework prišiel s kontajnerom AVI. Priniesol aj API, ktoré umožňuje vývojárom manipulovať a prehrávať video v ich vlastných aplikáciách a súbor nástrojov na prehrávanie a editovanie videa.

Knižnica pre kodek v tomto frameworku je implementovaná pomocou driveru (formát .dll) a je nainštalovaná pomocou súboru .inf alebo pomocou inštalátora. Oba tieto spôsoby inštalácie zapisujú hodnoty do registrov operačného systému. Na pridanie do 64-bitovej verzie Windows 7 je potrebné ešte túto knižnicu skopírovať do priečinku SysWOW64. Na vývoj bolo použité prostredie MS Visual Studio 2015 vzhľadom na to, že novšie verzie tento framework už nepodporujú. Funkcie volané týmto frameworkom sú rovnaké ako pri FFmpeg. Rozhranie vyzerá nasledovne.

```
LRESULT WINAPI DriverProc(
        DWORD dwDriverId,
        HDRVR hdrvr,
        UINT msg,
        LONG lParam1,
        LONG lParam2)
{
        codec_context *ctx = (codec_context *)dwDriverId;

        switch(msg)
        {
                case ICM_COMPRESS_QUERY:
                        return CompressQuery(ctx, (BITMAPINFO *)lParam1,
                         (BITMAPINFO *)lParam2);

                case ICM_COMPRESS_BEGIN:
                        return CompressQuery(ctx, (BITMAPINFO *)lParam1,
                         (BITMAPINFO *)lParam2);

                case ICM_COMPRESS_GET_FORMAT:
                        return CompressGetFormat((BITMAPINFO *)lParam1,
                         (BITMAPINFO *)lParam2);

                case ICM_COMPRESS_GET_SIZE:
                        return CompressGetSize((BITMAPINFO *)lParam1,
                         (BITMAPINFO *)lParam2);

                case ICM_COMPRESS:
                        return Compress(ctx, (ICCOMPRESS*)lParam1,
                         (DWORD)lParam2);
        }
}
```

Video for Windows bolo použité s aplikáciou VirtualDub, ktorá používa tento framework na prácu s videom. Na nasledujúcom obrázku 3.3 sú zobrazené základné informácie ohľadom kodeku - jeho FourCC kód a názov driveru použitého pri inštalácii. V kodeku nie je možné meniť použitý prediktor, vždy treba celý driver nanovo kompilovať a inštalovať.

V prílohe pri zdrojových kódoch sa nachádza .inf súbor pomocou ktorého je možné tento kodek nainštalovať.



Obrázok 3.3: Implementovaný kodek viditeľný v programe VirtualDub.

3.2 Implementácia

Pre framework FFmpeg je kodek implemtovaný v jazyku C. Voľba tohto jazyka zohľadňuje fakt, že počet operácií je veľmi vysoký a kodek má veľké požiadavky na výkon. Pre Video for Windows bol použitý jazyk C++, ktorý volá funkcie napísané v jazyku C. Bolo využité vývojárske prostredie MS Visual Studio 2015.

Jadro kodeku je multiplatformné, teda je ho možné preniesť na akýkoľvek operačný systém a zabudovať do frameworku. Na zmenu použitého prediktora je potrebné zmeniť posledný parameter funkcie *predict()*, na hodnotu, ktorá je daná v súbore *mkpcore* a korešponduje s konkrétnym prediktorom.

V tejto kapitole bol popísaný návrh kompresného algoritmu, jeho jednotlivých komponentov, ktoré budú ďalej testované s cieľom dosiahnutia čo najlepšieho kompresného pomeru. Ten bude predmetom porovnania s výsledkami kompresie kodekov Huffyuv a FFV1 v nasledujúcej kapitole. Ďalej tu boli popísané jednotlivé frameworky a návrh konkrétnej implementácie kodeku pre FFmpeg a Video for Windows.

Kapitola 4

Dosiahnuté výsledky

Táto kapitola sa zaoberá dosiahnutými výsledkami, kompresnými pomermi a zisteniami z práce s použitými technikami popísanými v predošlých kapitolách. Bolo experimentované s viacerými verziami farebných modelov, rôznymi typmi prediktorov s kontextami i bez nich, za účelom nájdenia najlepšej kombinácie pre bezstratovú kompresiu.

Testovacia vzorka videozáznamov bola zvolená z rôznych typov videí, vo vzorkách sa nachádzajú animované, hrané filmy, aj videá s veľkým množstvom strihov, zmien scén a videá, ktoré sú poškodené štvorcovými artefaktami po kompresii stratovým kodekom. Základné parametre jednotlivých vzoriek sú uvedené v tabuľke 4.1.

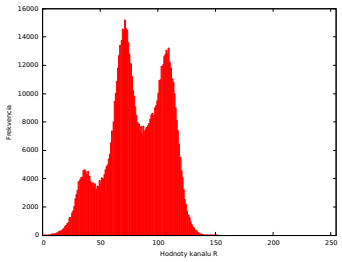
Č.	Názov videa	Rozlíšenie	Počet snímok	Trvanie [min:s]	Veľkosť [MB]
1	alien	854 × 480	2760	1:32	1628
2	back_to_the_future	854 × 480	1770	0:59	1055
3	dunham	854 × 480	3571	1:59	374
4	mash	854 × 480	3565	1:59	1926
5	prednaska	854 × 480	7170	2:59	3169
6	two_and_half_men	854 × 480	3600	2:00	2120
7	it_crowd	854 × 480	2070	1:09	1227
8	mash_scenes	854 × 480	1470	0:49	1051
9	scenes	854 × 480	4170	2:19	2463
10	animation	854 × 480	30	0:01	24
11	cars	1280 × 534	240	0:10	235
12	nemo	624 × 360	503	0:10	81
13	planes	1280 × 720	242	0:10	317
14	simpsons	854 × 480	2350	1:34	1750
15	tbbt_animation	854 × 480	3150	1:45	1226
16	toy_story	480 × 340	241	0:10	60
17	it_crowd_animation	854 × 480	2070	1:09	525
18	ice_age	854 × 480	3568	1:59	2106

Tabulka 4.1: Základné parametre testovacích vzoriek.

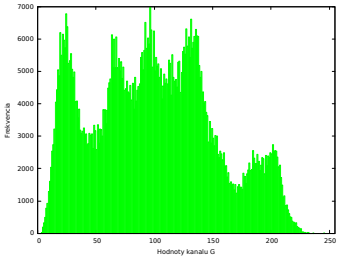
Vzorky 1 až 7 sú hrané filmy, ktoré obsahujú málo zmien scén. Nasledujú vzorky 8 a 9, ktoré sú hrané, scéna sa mení veľmi často, približne každých 5 sekúnd. Treťou skupinou sú vzorky 10 až 18, sú to animované filmy.

4.1 Výber vhodného farebného modelu

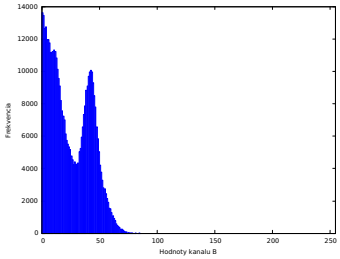
Pred kompresiou je obraz vo formáte 24-bitového RGB, kde rozloženie hodnôt nie je sústredené k žiadnej z hodnôt, čo je vidno z analýzy histogramov jednotlivých kanálov zobrazených na obrázkoch 4.1, 4.2 a 4.3. Tak isto ani transformácia farebného modelu tieto hodnoty nesústredí do jednej oblasti, čo vyplýva z obrázkov 4.4, 4.5, 4.6, avšak dekoreluje vzťah medzi jednotlivými zložkami. Nasledujúce grafy popisujú rozdelenie hodnôt farebných kanálov pred a po transformácii farebného modelu.



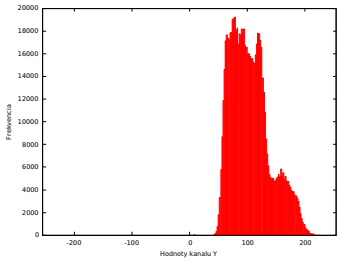
Obrázok 4.1: Kanál R.



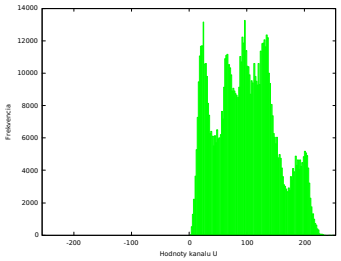
Obrázok 4.2: Kanál G.



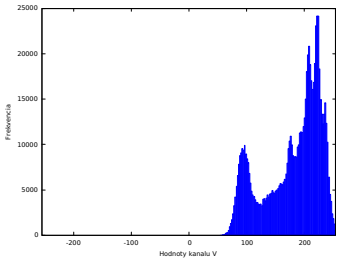
Obrázok 4.3: Kanál B.



Obrázok 4.4: Kanál Y.

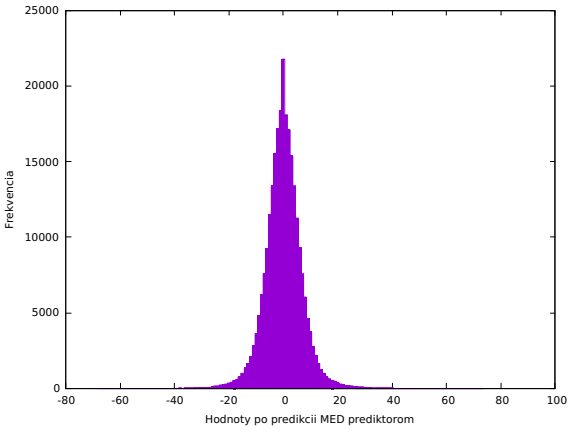


Obrázok 4.5: Kanál U.

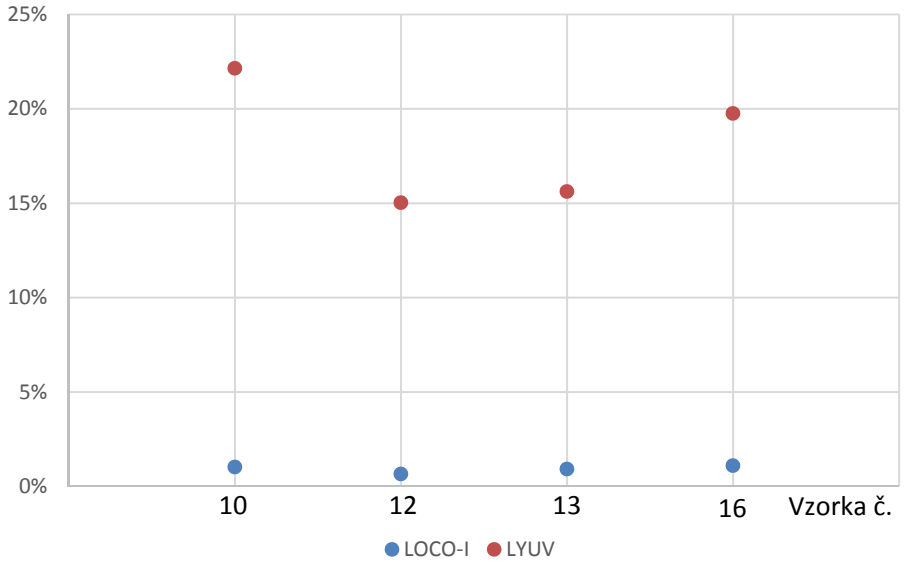


Obrázok 4.6: Kanál V.

Po aplikovaní vhodného prediktora sa hodnoty jednotlivých zložiek sústredia do okolia 0 ako vidno z obrázku 4.7, čo je výhodné pre entropický kodér, ktorý úsporne zaznamenáva často sa vyskytujúce hodnoty.



Obrázok 4.7: Histogram hodnôt po použití prediktora pre kanál Y.



Obrázok 4.8: Porovnanie veľkostí súborov oproti JPEG 2000 RCT v %. Na ose x sa nachádza číslo vzorky, ktorá bola meraná.

Bolo experimentované s farebnými transformáciami LOCO-I, LYUV a JPEG 2000 RCT. V teste bol použitý MED prediktor s kontextami a aritmetický kodér.

Vzorka č.	Názov	LOCO-I	JPEG 2000 RCT	LYUV
10	animation	6109	6047	7386
12	nemo	11741	11665	13418
13	planes	25050	24823	28700
16	toy_story	17581	17390	20824

Tabulka 4.2: Porovnanie veľkostí súborov v [kB] pri rôznych farebných modeloch.

Z výsledkov meraní uvedených v tabuľke 4.2 vyplýva, že z hľadiska kompresných pomerov je najvhodnejšou transformáciou JPEG 2000 RCT, za ňou nasleduje LOCO-I s minimálnymi rozdielmi. Veľkosť súboru po farebnej transformácii LOCO-I bola väčšia o 0,6 až 1,1 percenta oproti JPEG 2000 RCT. Poslednou farebnou transformáciou je LYUV, kde rozdiel veľkostí súborov predstavoval 15 až 22 percent. Tieto výsledky sú zobrazené v obrázku 4.8. Pre ďalšiu prácu bola zvolená farebná transformácia LOCO-I ako najlepší kompromis medzi jednoduchosťou (rýchlosťou) výpočtov a výsledkami kompresie, ktoré sú veľmi podobné ako pri použití najlepšieho farebného modelu JPEG 2000 RCT.

4.2 Výber vhodného prediktora

Táto kapitola popisuje výsledky dosiahnuté použitím rôznych prediktorov a porovnáva vhodnosť ich použitia na jednotlivé vzorky videozáznamov uvedených v tabuľke 4.1. Na porovnanie účinnosti jednotlivých prediktorov boli výsledky kompresie porovnávané s vý-

Č.	Názov videa	MED	Paeth	GAP	Huffyuv kodek	FFV1 kodek
1	alien	**396,0**	555,9	918,8	701,6	210,9
2	back_to_the_future	**409,5**	574,2	973,3	551,4	230,4
3	dunham	**151,5**	213,5	374,1	194,7	86,6
4	mash	**548,7**	742,0	1419,0	847,5	285,5
5	prednaska	**763,7**	1026,0	2499,0	1329,0	422,6
6	two_and_half_men	**609,7**	846,5	1576,0	912,0	304,1
7	it_crowd	**485,6**	734,9	1332,0	666,0	256,5
8	mash_scenes	**296,6**	396,1	711,6	473,5	161,6
9	scenes	**856,5**	1245,0	2355,0	1132	452,7
10	animation	**6,6**	9,9	16,7	10,8	3,0
11	cars	**38,1**	67,9	171,1	101,0	21,3
12	nemo	**12,4**	20,9	59,6	35,7	5,6
13	planes	**26,7**	51,9	180,2	125,4	12,1
14	simpsons	**557,0**	833,2	1464,0	881,5	257,3
15	tbbt_animation	**398,6**	580,3	1074,0	698,7	208,4
16	toy_story	**18,5**	28,8	59,4	35,6	9,6
17	it_crowd_animation	**138,3**	198,1	485,4	236,4	57,8
18	ice_age	**864,9**	1192,0	2182,0	1064,0	471,7

Tabulka 4.3: Veľkosti videí po kompresii rôznymi prediktormi bez použitia kontextov v [MB].

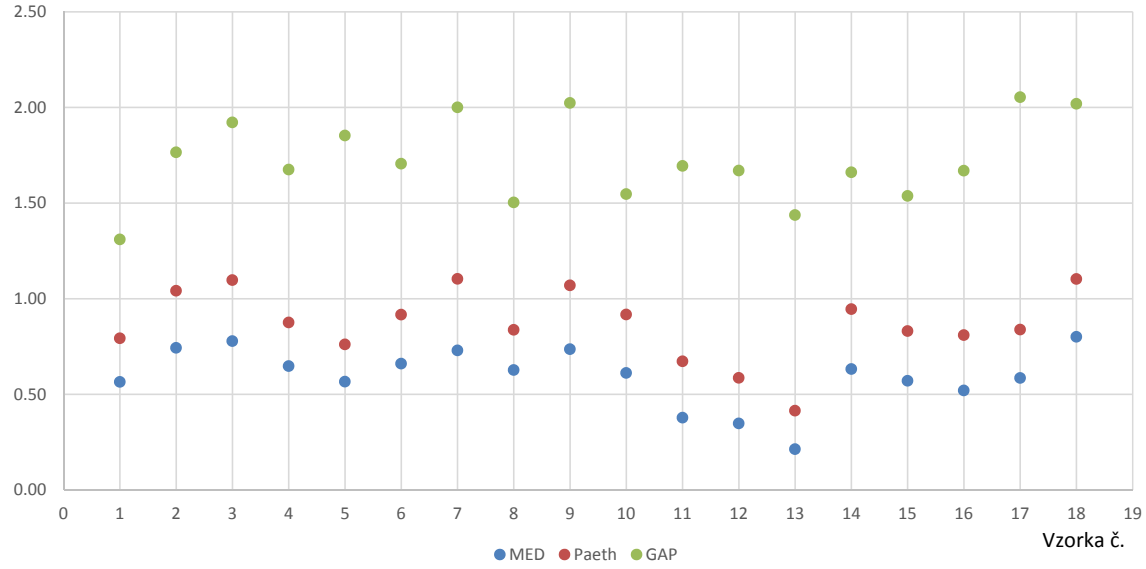
sledkami kodekov bežne používaných na bezstratovú kompresiu Huffyuv a FFV1. Najlepšie dosiahnuté výsledky sú označené v tabuľkách tučným písmom.

4.2.1 Porovnanie prediktorov bez použitia kontextov

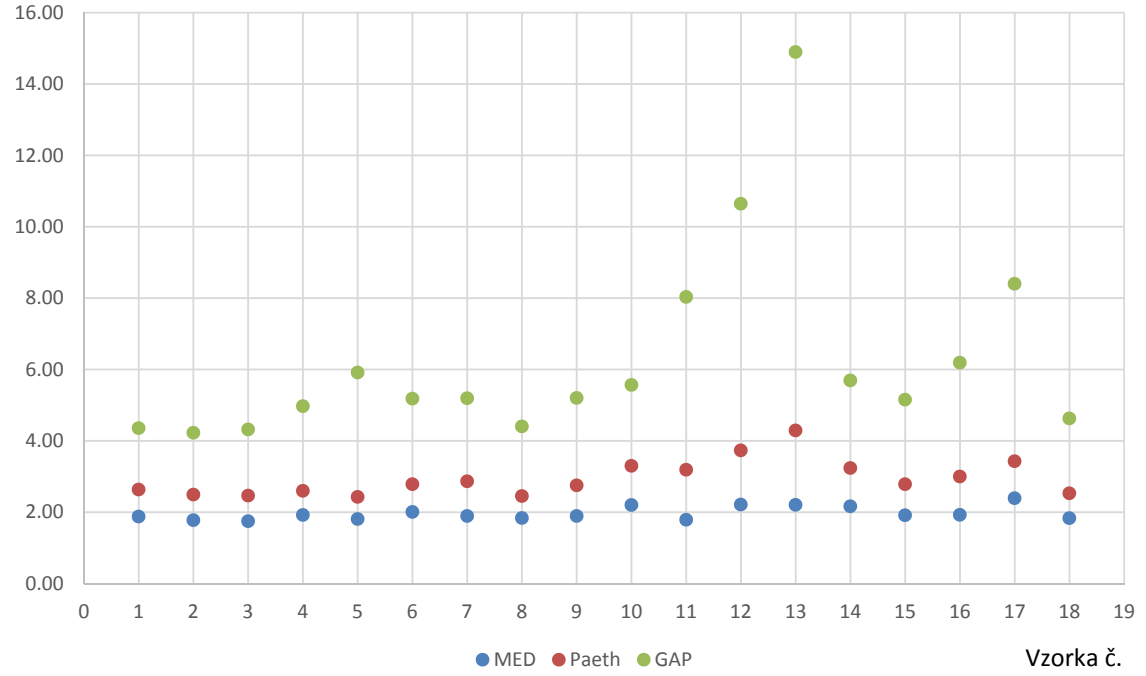
Bola porovnávaná účinnosť kompresie prediktorov MED, Paeth a GAP, ktoré sú podrobne popísané v kapitole 2.3. Výsledky merania sú uvedené v tabuľke 4.3

Vo všetkých prípadoch najlepší kompresný pomer zo všetkých implementovaných prediktorov bez použitia kontextov dosahuje prediktor MED. Na všetkých vzorkách bol dosiahnutý lepší kompresný pomer ako dosiahol kodek Huffyuv. Aj prediktor Paeth bez použitia kontextov dosiahol na väčšine vzoriek lepší kompresný pomer ako kodek Huffyuv. Poradie dosiahnutých veľkostí kompresných pomerov je v priemere nasledovné:

1. FFV1 kodek
2. MKP kodek s MED prediktorom
3. MKP kodek s Paeth prediktorom
4. Huffyuv kodek
5. MKP kodek s GAP prediktorom



Obrázok 4.9: Porovnanie dosiahnutých kompresných pomerov s kodekom Huffyuv. Na ose x sa nachádza číslo vzorky, ktorá bola meraná. Na vertikálnej ose je pomer komprimovaných súborov MKP/Huffyuv. Výsledok menší ako 1 znamená prekonanie kodeku Huffyuv.



Obrázok 4.10: Porovnanie dosiahnutých kompresných pomerov s kodekom FFV1. Na ose x sa nachádza číslo vzorky, ktorá bola meraná. Na vertikálnej ose je pomer komprimovaných súborov MKP/FFV1. Výsledok menší ako 1 znamená prekonanie kodeku FFV1.

Ako vidno na obrázku 4.9 MED prediktor v porovnaní s Huffyuv kodekom dosahoval kompresný pomer v priemere o 41% lepší. Paeth prediktor dosahoval v priemere lepší kompresný pomer o 13%, GAP bol horší v priemere o 72%.

Ako vidno na obrázku 4.10 kodek FFV1 sa nepodarilo prekonať, MED prediktor bol v priemere horší o 97%, Paeth prediktor o 194% a GAP o 528%. V prípade hraných filmov (vzorky 1 až 9) boli výsledky implementovaného kodeku v porovnaní s FFV1 lepšie ako pri animovaných záznamoch (vzorky 10 až 18).

4.2.2 Porovnanie prediktorov s kontextami

Testovacou vzorkou sú rovnaké videozáznamy ako v predošlej podkapitole 4.2.1. Rozdielom je použitie prediktorov s kontextami, ktoré sú určované v závislosti od zvolenej vetvy v prediktore. Vo všetkých prípadoch je kompresný pomer na rovnakej vzorke videa o niekoľko percent lepší ako bez ich použitia. Vzhľadom na minimálne zvýšené pamäťové nároky oproti variante bez kontextov je oveľa výhodnejšie použiť čo i len malý počet kontextov. Najlepšie dosiahnuté výsledky sú označené v tabuľke 4.4 tučným písmom.

Č.	Názov videa	MED	Paeth	GAP	Huffyuv kodek	FFV1 kodek
1	alien	**376,4**	508,9	829,6	701,6	210,9
2	back_to_the_future	**390,9**	538,1	869,4	551,4	230,4
3	dunham	**146,1**	202,4	323,9	194,7	86,6
4	mash	**517,0**	682,6	1220,0	847,5	285,5
5	prednaska	**717,9**	925,7	2128,0	1329,0	422,6
6	two_and_half_men	**572,5**	777,5	1373,0	912,1	304,1
7	it_crowd	**462,0**	690,9	1181,0	666,0	256,5
8	mash_scenes	**281,6**	366,2	606,8	473,5	161,6
9	scenes	**816,0**	1166,0	2036,8	1164,0	452,7
10	animation	**6,1**	9,0	14,6	10,8	3,0
11	cars	**35,3**	60,0	145,5	101,1	21,3
12	nemo	**11,7**	19,4	51,9	35,7	5,6
13	planes	**25,0**	46,2	150,4	125,4	12,1
14	simpsons	**509,7**	761,1	1295,2	881,5	257,3
15	tbbt_animation	**379,9**	545,4	947,5	698,7	208,4
16	toy_story	**17,6**	27,2	52,7	35,6	9,6
17	it_crowd_animation	**129,9**	182,0	411,0	236,4	57,8
18	ice_age	**820,6**	1114,0	1951,0	1081,0	471,7

Tabulka 4.4: Veľkosti videí po kompresii rôznymi prediktormi s použitím kontextov v [MB].

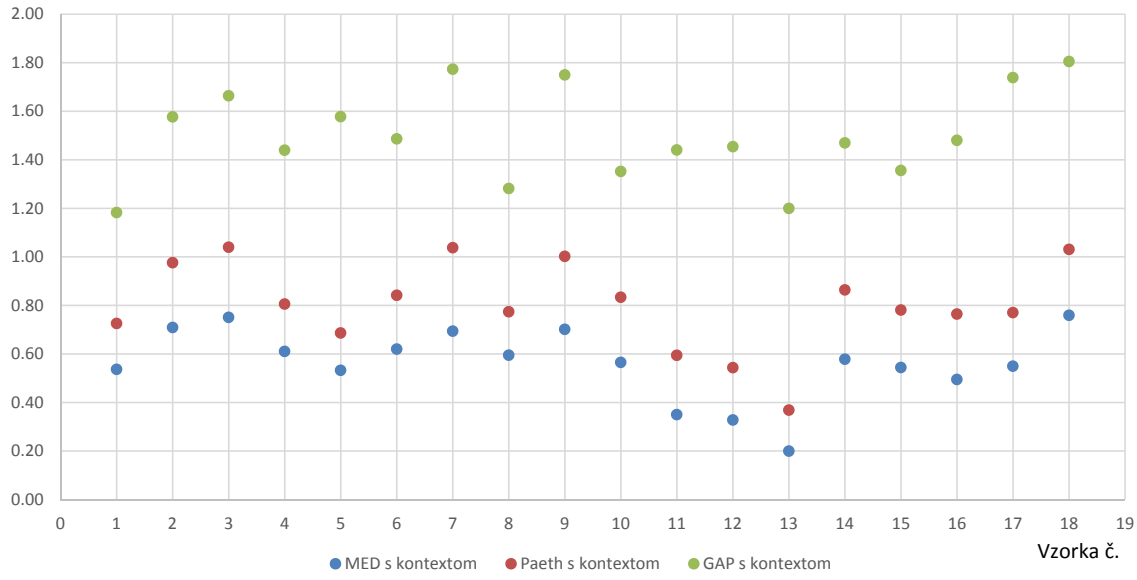
Poradie dosiahnutých kompresných pomerov je nasledovné:

1. FFV1 kodek
2. MKP kodek s MED prediktorom a tromi kontextami
3. MKP kodek s Paeth prediktorom a tromi kontextami
4. Huffyuv kodek

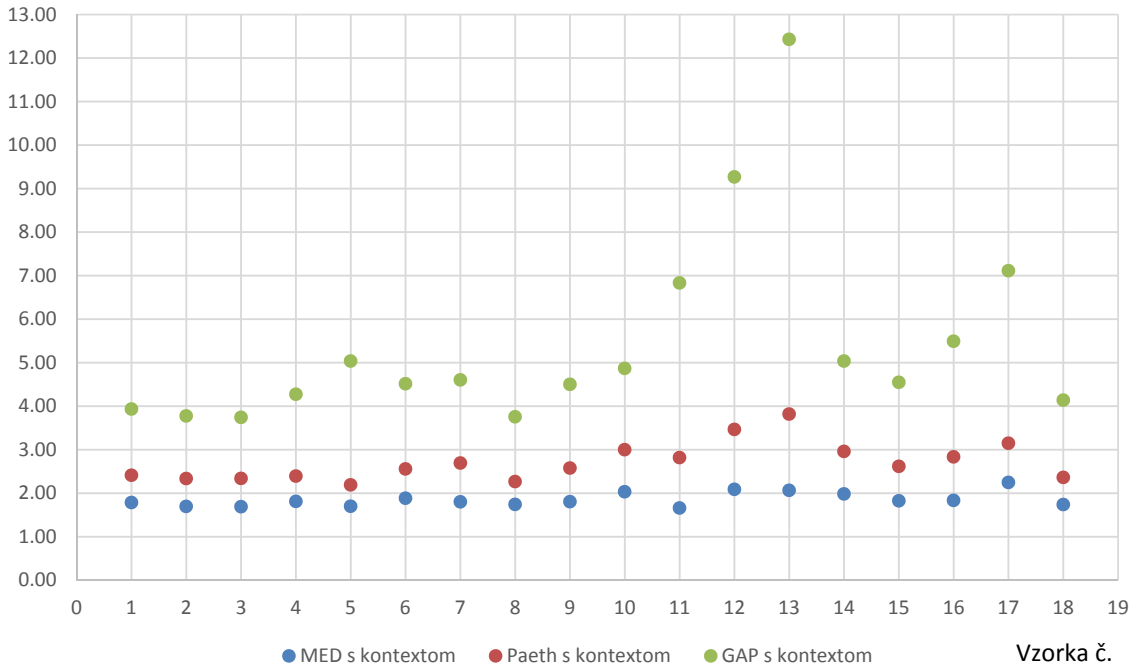
5. MKP kodek s GAP prediktorom a siedmimi kontextami

Navrhovaný MKP kodek dosahuje najlepšie výsledky kompresie vo verzii s MED prediktorom s kontextami a dosiahol lepší kompresný pomer ako Huffyuv, avšak tento výsledok je stále horší ako v súčasnosti najpoužívanejší bezstratový kodek FFV1. Kodek MKP používa pre každú vetvu prediktora jeden kontext, kodek FFV1 používa veľké množstvo kontextov, ktorými dosahuje výrazne lepšie kompresné pomery. Kodek MKP dosahoval horšie výsledky pri kompresii videozáznamov poškodených stratovou kompresiou (vzorky 3 a 18), ktoré obsahovali drobné štvorčekové artefakty. FFV1 dosiahol najlepšie kompresné pomery pri kompresii animovaných filmov, čo vidno v pravej strane obrázku 4.12. Do finálnej verzie kodeku bol použitý MED prediktor s tromi kontextami.

Ako vidno na obrázku 4.11 MED prediktor v porovnaní s Huffyuv kodekom dosahoval kompresný pomer v priemere o 44% lepší. Paeth prediktor dosahoval v priemere lepší kompresný pomer o 20%, GAP bol horší v priemere o 50%.



Obrázok 4.11: Porovnanie dosiahnutých kompresných pomerov s kodekom Huffyuv. Na ose x sa nachádza číslo vzorky, ktorá bola meraná. Na vertikálnej ose je pomer komprimovaných súborov MKP/Huffyuv. Výsledok menší ako 1 znamená prekonanie kodeku Huffyuv.



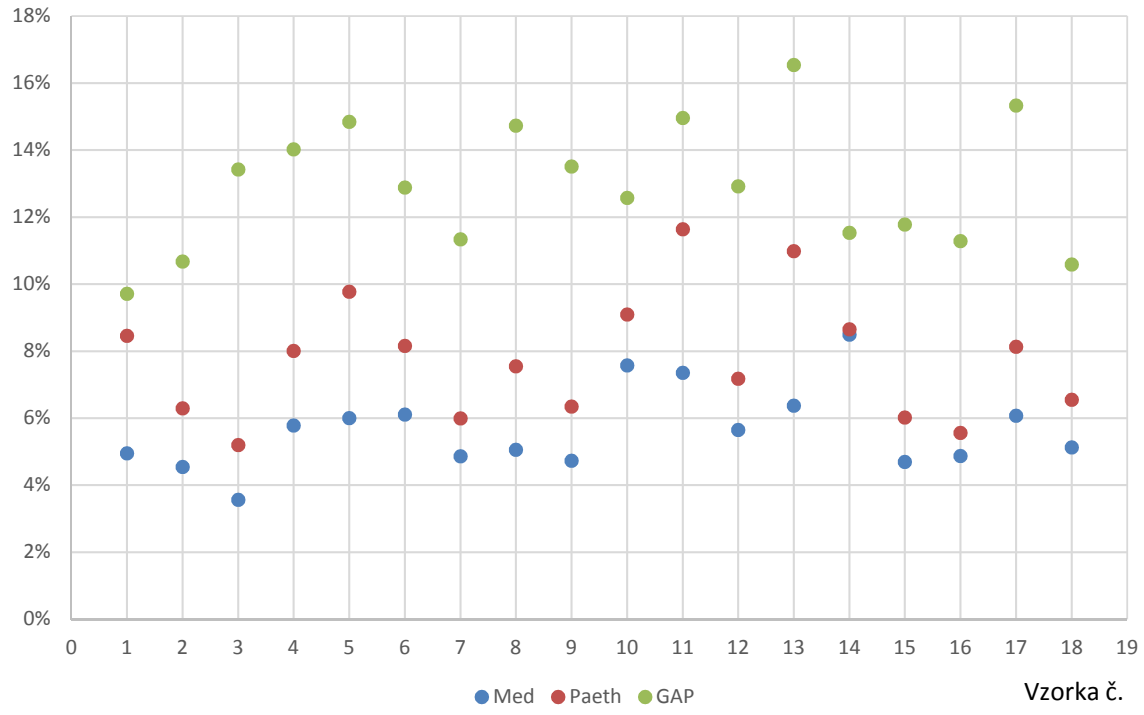
Obrázok 4.12: Porovnanie dosiahnutých kompresných pomerov s kodekom FFV1. Na ose x sa nachádza číslo vzorky, ktorá bola meraná. Na vertikálnej ose je pomer komprimovaných súborov MKP/FFV1. Výsledok menší ako 1 znamená prekonanie kodeku FFV1.

Ako vidno na obrázku 4.12 kodek FFV1 sa nepodarilo prekonať, MED prediktor bol v priemere horší o 85%, Paeth prediktor o 171% a GAP 444%. V prípade hraných filmov (vzorky 1 až 9) boli výsledky implementovaného kodeku v porovnaní s FFV1 lepšie ako pri animovaných záznamoch (vzorky 10 až 18).

Lepšie výsledky kompresie boli vždy dosiahnuté vo verzii kompresného algoritmu s kontextami. Podľa výsledkov analýzy uvedených v tabuľke 4.5 aj použitím malého počtu kontextov je možné výrazne upraviť kompresný pomer. V tabuľke sú vyznačené vždy najnižšie (žltou) a najvyššie (zelenou) zmeny kompresného pomeru použitím kontextu v percentách. Pri použití MED prediktora s tromi kontextami sa zlepšenie kompresného pomeru pohybuje okolo 5%, pri použití Paeth prediktora s tromi kontextami sa zlepšenie kompresného pomeru pohybuje okolo 7%, najvyššie zlepšenie bolo dosiahnuté GAP prediktorom so siedmimi kontextami a pohybovalo sa okolo 12%.

Č.	Názov videa	MED	Paeth	GAP
1	alien	4,95	8,45	9,71
2	back_to_the_future	4,54	6,29	10,68
3	dunham	3,56	5,20	13,42
4	mash	5,78	8,01	14,02
5	prednaska	6,00	9,78	14,85
6	two_and_half_men	6,10	8,15	12,88
7	it_crowd	4,86	5,99	11,34
8	mash_scenes	5,06	7,55	14,73
9	scenes	4,73	6,35	13,51
10	animation	7,58	9,09	12,57
11	cars	7,35	11,63	14,96
12	nemo	5,65	7,18	12,92
13	planes	6,37	10,98	16,54
14	simpsons	8,49	8,65	11,53
15	tbbt_animation	4,69	6,01	11,78
16	toy_story	4,86	5,56	11,28
17	it_crowd_animation	6,07	8,13	15,33
18	ice_age	5,12	6,54	10,59

Tabulka 4.5: Porovnanie zlepšenia kompresného pomeru v [%] po použití kontextov.



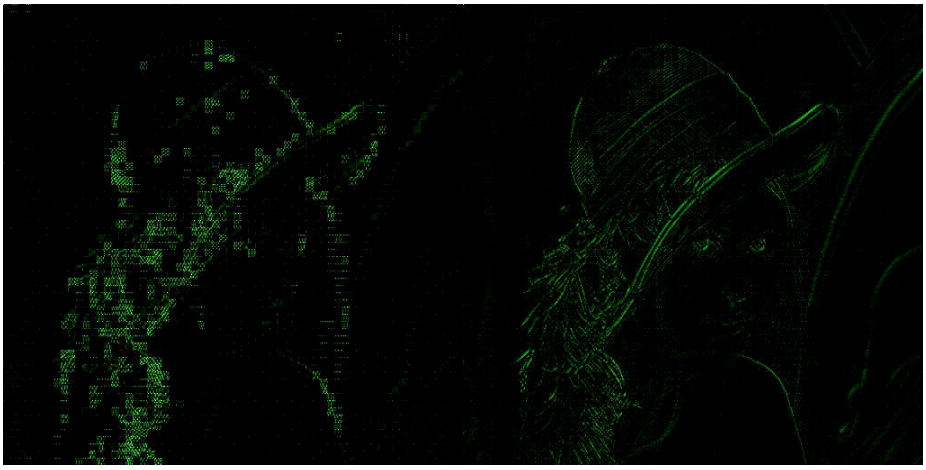
Obrázok 4.13: Zlepšenie kompresného pomeru použitím kontextov v [%]. Na ose x sa nachádza číslo vzorky, ktorá bola meraná.

Z uvedených výsledkov kompresných pomerov a tiež zo špecifikácie kodeku FFV1 vyplýva, že kontexty dokážu výraznou mierou prispieť k zlepšeniu výsledkov, ktoré sa týkajú kompresných pomerov. FFV1 kodek používa na dosiahnutie dobrých kompresných pomerov vysoký počet kontextov. Tento prístup zvyšuje výpočtovú náročnosť a znižuje rýchlosť tohto kodeku, napriek tomu, že využíva paralelné spracovanie jednotlivých častí snímky.

Z uvedených meraní vyplýva, že ďalšie zlepšenie výsledkov kompresného pomeru je možné dosiahnuť pridaním ďalších kontextov do MKP kodeku a zároveň nevyhnutnosť optimalizovať výpočtovú rýchlosť využitím paralelného spracovania údajov.

4.3 Videá poškodené kompresiou

Táto podkapitola sa venuje popisu správania sa MED prediktora v prípade, že video je poškodené stratovou kompresiou, ktoré prediktor detekuje ako hrany. V tomto prípade sa jedná o vzorky 3 a 18. Na obrázku 4.14 a 4.15 je názorná ukážka ako MED prediktor detekuje hrany artefaktov ako dodatočné hrany v snímke. Z toho vyplýva horší kompresný pomer, keďže sa rozdiely predikovanej a reálnej hodnoty menej blížia k 0, čo je nevýhodné pre aritmetický kodér. Pri oboch vzorkách prišlo k zhoršeniu kompresného pomeru o približne 20% v porovnaní s priemernou hodnotou kompresného pomeru.



Obrázok 4.14: Poškodená snímka.

Obrázok 4.15: Pôvodná snímka.

Poškodenie štvorcovými artefaktami nemusí vždy vyústiť do horšieho kompresného pomeru. V prípade, že stratová kompresia je príliš agresívna, v snímke sa nachádza viac hrán, ale hodnoty pixelov v rámci štvorcového artefaktu sú rovnaké. Vďaka tomu je stále možné dosiahnuť dobrý kompresný pomer. Rovnako je možné usúdiť, že pokiaľ štvorcové artefakty sú prítomné, ale nie výrazné a informácia v rámci artefaktu je uchovaná vo väčšej miere (stratová kompresia nie príliš agresívna), tak pri danom rozlíšení je kompresný pomer horší ako pri videu, ktoré nebolo stratovo komprimované.

V tejto kapitole bola vyhodnotená výkonnosť jednotlivých stavebných blokov navrhovaného kodeku z hľadiska kompresného pomeru. Najlepšie výsledky dosahoval kodek s farebným modelom LOCO-I, MED prediktorom s kontextami a aritmetickým kodérom. V nasledujúcej kapitole sú sumarizované výsledky práce.

Kapitola 5

Záver

Táto práca sa zaoberala jednotlivými komponentami potrebnými na implementáciu bezstratovej kompresie videa, ktoré boli použité na vytvorenie bezstratového videokodeku. V jednotlivých kapitolách boli popísané základné stavebné bloky, ich funkčnosť, možnosti a typy implementácie.

Na základe uvedenej teórie bol implementovaný bezstratový kodek nazvaný MKP. Používa farebný model LOCO-I, MED prediktor s kontextami a aritmetický kóder. Pri jeho vytváraní bolo experimentované s prediktormi MED, Paeth, GAP s použitím jedného, troch alebo siedmych kontextov. Tento kodek bol testovaný v šiestich rôznych variantách a porovnávaný s aktuálne používanými bezstratovými kodekmi Huffyuv, ktorý vyniká rýchlosťou kompresie a FFV1, ktorý dosahuje výborné kompresné pomery. Vo všeobecnosti použitie kontextov prinieslo zlepšenie kompresného pomeru o 3 až 16%. Najviac z tejto metódy profitoval GAP prediktor, najmenej MED. Celkovo najlepšie výsledky kompresných pomerov dosiahol MED prediktor s troma kontextami. Výsledky dosahované týmto video kodekom boli porovnávané len na základe kompresného pomeru.

Kompresné pomery jednotlivých verzií kodekov boli merané na reprezentatívnej vzorke videí rôznych typov: animovaných, hraných, s veľkým množstvom zmien scén a takých, ktoré sú poškodené stratovou kompresiou a obsahujú štvorcové artefakty. Tabuľky porovnávajúce kodeky Huffyuv a FFV1 s výslednými kompresnými pomermi sú uvedené v podkapitolách 4.2.1, 4.4. Implementovaný kodek vo verzii s MED prediktorom s kontextami vo všetkých prípadoch výrazne prekonáva kodek Huffyuv, v priemere o 44%, ale v porovnaní s kodekom FFV1 je výsledný súbor v priemere dvakrát taký veľký.

Táto konkrétna varianta kodeku bola implementovaná do frameworkov FFmpeg pre operačný systém Linux a Video for Windows ako ovládač pre operačný systém Microsoft Windows. Ďalšie možnosti rozšírenia práce zahŕňajú:

- Použitie kompresie obrazu pomocou vlnkovej transformácie s využitím jej celočíselnej implementácie akou je napríklad vlnka Daubechies 5/3 s použitím optimalizovaného liftingového algoritmu. Tento spôsob kompresie využíva štandard JPEG 2000.

- Využitie väčšieho počtu kontextov – inšpirované kompresnými výsledkami kodeku FFV1. Navrhujem otestovať použitie vysokého počtu kontextov (rádovo stovky) a následne subkontexty (rádovo desiatky).

- Implementovaný kompresný algoritmus nebol optimalizovaný na rýchlosť kompresie. Jeho výkon v tejto oblasti je možné zvýšiť paralelizovaním algorimu ako napr. FFV1, ktorý rozdeľuje snímku na makrobloky (slices), ktoré spracúva paralelne. Pri tomto

prístupe je tiež potrebné hľadať vhodný kompromis medzi želaným kompresným pomerom a rýchlosťou kompresie, pretože zvyšovanie počtu makroblokov môže priniesť zvýšenie rýchlosti kompresie a zároveň zhoršenie kompresného pomeru.

- Experimentovať s presnoťou predikcie AGSP prediktora a porovnať dosiahnuté hodnoty kompresných pomerov.
- Implementácia odhadu a kompenzácie pohybu je ďalšou oblasťou, kde by sa dal zvýšiť kompresný pomer, ale znížila by sa rýchlosť kompresného algoritmu v dôsledku celkového zvýšenia výpočtovej náročnosti algoritmu. V bežne používaných bezstratových kodekoch je tento prístup (inter-frame) skôr výnimkou ako pravidlom. Takáto implementácia bezstratovej kompresie je vhodná len pre videosekvencie, ktoré obsahujú len málo a pomalých zmien.

Literatura

[1] Bodden, E.; Clasen, M.; Kneis, J.: Arithmetic coding revealed-a guided tour from theory to praxis. 2007, Sable Research Group, McGill University, Tech. správa Sable Technical Report 2007-5.

[2] Halliday, D.; Resnick, R.; Walker, J.: *Fyzika*, ročník 2. VUTIUM, druhé vydání, 2013, ISBN 978-80-214-4123-1.

[3] Howard, P. G.; Vitte, J. S.: Analysis of Arithmetic Coding for Data Compression. *Information Processing and Management*, 1992, ISSN 0306-4573.

[4] Niedermayer, M.: FFV1 Video Codec Specification. [Online; navštívené 6.5.2016]. URL https://www.ffmpeg.org/~michael/ffv1.html

[5] Nishad, P.; Manicka, C.: Various Colour Spaces and Colour Space Conversion Algorithms. *Journal of Global Research in Computer Science*, 2013, ISSN 2229-371X.

[6] Rabbani, M.: The JPEG2000 Still-Image Compression Standard. *IEEE Signal Process Magazine*, 2001, ISSN 1053-5888.

[7] Sayood, K.: *Introduction to data compression*. Elsevier, 2012, ISBN 978-0-12-415796-5.

[8] Tang, H.; Kamata, S.: An Efficient Encryption and Compression System for Grayscale and Color Images. *IEICE Transactions on Information and Systems*, 2006, ISSN 0916-8532.

[9] Tischer, P. E.; Worley, R. T.; Maeder, A. J.; aj.: Context-based Lossless Image Compression. *Computer Journal*, 1992, ISSN 1460-2067.

[10] Vleuten, R.; Egner, S.: Lossless and Fine-Granularity Scalable Near-Lossless ColorImage Compression. *Proceedings - Data Compression Conference*, 2002, ISSN 1068-0314.

[11] Weinberger, M.; Seroussi, G.; Sapiro, G.: The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS. 1998, HPL-98-193.

[12] Wheeler, F.: Adaptive arithmetic coding source code. [Online; navštívené 16.12.2016]. URL http://www.fredwheeler.org/ac/

[13] Willen, I. H.; Neal, R. M.; Cleary, J. G.: Arithmetic coding for data compression. *Communications of the ACM*, 2012, ISSN 0001-0782.

Prílohy

Príloha A

Obsah DVD

- Zdrojové kódy kodeku pre framework FFmpeg
- Zdrojové kódy kodeku pre framework Video for Windows (MS Visual Studio)
- Dokumentácia k programu
- Zdrojový kód dokumentácie v LaTeX
- Sada testovacích videí