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

IMPLEMENTACE JPEG 2000

JPEG 2000 IMPLEMENTATION

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **ADAM ZLATOHLÁVEK**
AUTHOR

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

BRNO 2017

Abstrakt

Tato práce se zaobírá novodobým kompresním řetězcem JPEG 2000. První část je rozborem technik používaných v základní části standardu a jejich možností v kompresním formátu. Druhá část se věnuje implementaci kompresního řetězce z pohledu předzpracování, až k přivedení výstupu do vlastního formátu. Závěrem práce je provedeno srovnání s aktuálními implementacemi z paměťového a výkonnostního hlediska.

Abstract

The aim of this thesis is to propose a image compression methods in JPEG 2000. This consists of description of techniques used in base level stadard and analyze options in encoding process. The goal is to create compression process of input image from preprocessing to own output format. In the end of paperwork are presented results of own implementation in memory consuptions and encoding performance.

Klíčová slova

JPEG 2000, komprese obrazu, obrazový formát, diskrétní vlnková transformace, EBCOT, ztrátová komprese, vlnka, aritmetický kodér

Keywords

JPEG 2000, image compression, image format, discrete wavelet transform, EBCOT, lossy compression, wavelet, arithmetic coder

Citace

ZLATOHLÁVEK, Adam. *Implementace JPEG 2000*. Brno, 2017. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. David Bařina, Ph.D.

Implementace JPEG 2000

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Davida Bařiny Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Adam Zlatohlávek
17. května 2017

Poděkování

Rád bych poděkoval vedoucímu práce doktoru Davidu Bařinovi, který poctivě a trpělivě dohlížel nad vypracováním práce.

Obsah

Kapitola 1

Úvod

JPEG je obrazový formát, který přináší podporu pro ztrátovou i bezeztrátovou kompresi bez sebevětší změny postupu kódování. Jádrem kodeku je diskrétní vlnková transformace. Proto pro vybudování kvalitního kodéru a dekodéru je nutné navrhnout efektivní vlnkou transformaci. Významným milníkem je nutnost odladit tuto část a zajistit napojení na zbývající enkódovací proces.

Cílem práce je seznámit se s existujícími řešeními enkodéru, vyzkoušet si kompresní možnosti laděni parametrů a implementovat vlastní kompresní řetězec JPEG 2000 v jazyce C. Nakonec je implementace otestována na výkonnostní a paměťové nároky a srovnána s existujícími řešeními. Vlastní řešení nabízí nástroj pro enkódování, který připraví obraz na zpracování samotným jádrem kodeku, DWT. Po provolání transformace nad jednotlivými komponentami obrazu jsou vlnkové koeficienty zakódovány pomocí aritmetického kodéru. Výstupní datový proud je strukturován do paketů, které jsou přivedeny do výstupního souboru.

Kapitola 2

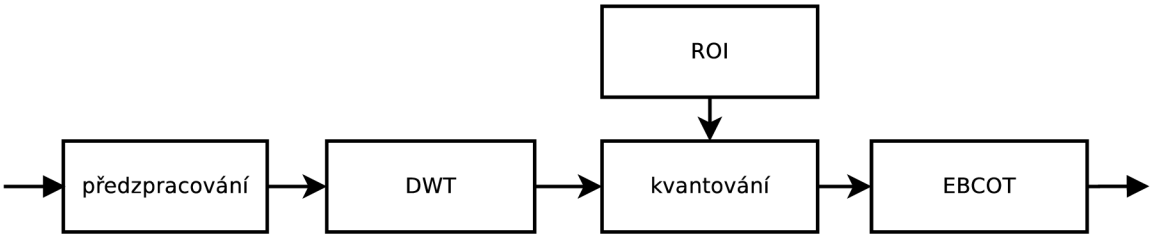
JPEG 2000

JPEG 2000 je mezinárodní standard pro obrazovou kompresi, který byl vytvořen pod dohledem společenstvím ISO/IEC. První a také základní část standardu je popsána v dokumentu ISO/IEC 15444-1 [5]. Jedná se o nástupce známého obrazového formátu JPEG z roku 1992, který je dnes využíván především ke ztrátové kompresi.

Průběhem času svět technologií a konzumního trhu prošel evolucí a bylo žádoucí přizpůsobit možnosti pro přenos obrazu skrz internet. Významným milníkem je použití metody nazvané diskrétní vlnková transformace (DWT). Tento koncept umožňuje ztrátovou i bezeztrátovou kompresi, efektivní enkódování při velmi nízkých datových tocích, škálovatelnost, interoperabilitu v síťovém provozu a především větší flexibilitu. Nevýhodou nového kódovacího algoritmu je vyšší komplexnost a vyšší potřeba výkonu při kompresi i dekompresi.

Celý standard je rozčleněn do 12 částí, kdy 7. část byla zavrhnuta. S každou částí jsou přidány nové funkce k základní části č. 1. V následujícím textu se zaměříme na popis základní části formátu JPEG 2000. Přehled všech ostatních částí je dostupný na [6].

Tak jako jiné standardy pro kompresi obrazových dat je JPEG 2000 vytvořen z pohledu dekodéru. Dekodér tak diktoval funkcionality pro enkodér. Implementace enkodéru byla pak řízena syntaxí datového toku.



Obrázek 2.1: Blokové schéma enkodéru JPEG 2000. Převzato ze zdroje [2]

Celý systém je rozdělen do 5 fází. V blokovém schématu 2.1 můžeme vidět, jak jsou propojeny jednotlivé fáze. Nazýváme je (1) předzpracování obrazu (2) DWT (3) kvantování (4) ROI a (5) EBCOT. Na vstup předzpracování obrazu je přiveden zdrojový obraz ke kompresi. Na výstupu EBCOT je pak komprimovaný obraz. Jednotlivé části budou podrobně probrány v následujících sekcích.

2.1 Předzpracování obrazu

Předzpracování obrazu se skládá ze 3 volitelných podčástí: rozdělení na dlaždice, posunutí úrovní obrazu a multikomponentní transformace.

2.1.1 Rozdělení na dlaždice

První operací předzpracování je obdélníková teselace nebo také rozdělení na dlaždice. V tomto kroku se zdrojový obraz volitelně rozdělí na menší obdélníkové bloky, které se zároveň nepřesahují. Každý takový blok se nazývá dlaždice. Všechny dlaždice mají stejnou velikost kromě dlaždic při hranicích obrazu, které mají zbytkovou velikost po celočíselném dělení. Maximální velikost dlaždice je velikost celého obrazu, kdy v tomto případě bude přítomna pouze jedna dlaždice. Dlaždice obsahuje pixely ze zdrojového obrazu pro danou lokalitu danou dvěma body, a to počátečním a koncovým.

Protože jsou dlaždice zpracovávány nezávisle, mohou se objevit artefakty u hranic jednotlivých dlaždic. Tento šum je zapříčiněn zpracováním pixelů a jejího okolí. Čím jsou tedy dlaždice menší, objevuje se tak i více obrazových artefaktů a je tak i snížena kompresní efektivita oproti větší velikosti dlaždic. Zpracování bez obdélníkové teselace tedy nabízí lepší vizuální kvalitu. Nicméně čím jsou dlaždice větší, tím úměrně vzrůstají paměťové nároky. U hardwarových kodérů VLSI je paměť však omezena. Počítá se tak s určitou velikostí dlaždice např. 256 x 256 nebo 512 x 512 pixelů podle [1].

2.1.2 Posunutí úrovní obrazu

Ve zdrojových datech je obrázek uložen v pixelech. Každý pixel obsahuje určitý počet barevnonosných složek, které jsou reprezentovány jako bezznaménkové celočíselné datové typy. Pro matematické operace je výhodné použít transformaci každého čísla na dvojkový komplement. Cílem transformace je zajištění, aby zdrojová data měly dynamický rozsah soustředěný kolem nuly.

Všechny vzorky obrazu $I_i(x, y)$ kde index i je komponentou obrazu (nebo dlaždice) jsou posunuty odečtem druhé mocniny $S^i - 1$. S^i znázorňuje přesnost barevnonosné komponenty obrazu neboli bitovou hloubku. Celý vztah je uveden v rovnici 2.1. S posunem lze v LL podpásmu DWT tak jako v HH podpásmu aplikovat filtr horní propusti a filtru dolní propusti na stejné úrovni, jen v rozdílu znaménka.

$$I_i^{'}(x, y) \leftarrow I_i(x, y) - 2^{S^i - 1} \tag{2.1}$$

Pro obrazy, kde barevnonosné složky jsou reprezentována jako celá čísla se znaménkem, je dynamický rozsah již soustředěn kolem nuly, a tak není nutné provádět posunutí úrovní obrazu. Příkladem takových dat mohou být snímky z CT tomografie.

2.1.3 Multikomponentní transformace

Multikomponentní transformace odbourává rozdíly mezi různými reprezentacemi komponent v obrazu. Napomáhá k snížení nadbytečných informací a k zvýšení kompresního výkonu (eliminací rozdílných reprezetancí). Ve standardu nejsou barevné kanály brány jako barevnonosné informace, ale naopak jako komponenty obrazu. Transformace definována pouze pro první tři komponenty. Pro jednoduchost si lze představit první tři komponenty jako barevné kanály (R, G, B). Každá komponenta může nabývat různé bitové hloubky a rozdíl-

ného rozsahu. Nicméně multikomponentní transformace počítá s prvními 3 komponentami o stejné bitové hloubce a taktéž rozsahu.

První část standardu JPEG 2000 [5] podporuje dvě rozdílné transformace. (1) reverzivní barevná transformace (RCT) a (2) ireverzivní barevná transformace (ICT). RCT lze použít pro ztrátovou i bezeztrátovou kompresi. V bezeztrátové variantě lze použít Forward RCT, která lze přesně rekonstruovat pomocí Inverse RCT do původní podoby. V ztrátové variantě je ztrátovost zapříčiněna kvantizačním krokem komprese, nikoliv samotným RCT převodem.

Dopředná RCT:

$$\begin{aligned} Y_r &= \left\lfloor \frac{R + 2G + B}{4} \right\rfloor \\ U_r &= B - G \\ V_r &= R - G \end{aligned} \tag{2.2}$$

Inverzní RCT:

$$\begin{aligned} G &= Y_r - \left\lfloor \frac{U_r + V_r}{4} \right\rfloor \\ R &= V_r + G \\ B &= U_r + G \end{aligned} \tag{2.3}$$

ICT je možné aplikovat pouze na ztrátovou kompresi, protože k převodu se používají racionální koeficienty jako vážené parametry v transformační matici. Při počítání s jak je uvedeno v rovnicích pro dopřednou ICT 2.4 a zpětnou ICT 2.5.

Dopředná ICT:

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0,299000 & 0,587000 & 0,114000 \\ -0,168735 & -0,331264 & 0,500000 \\ 0,500000 & -0,418688 & -0,081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2.4}$$

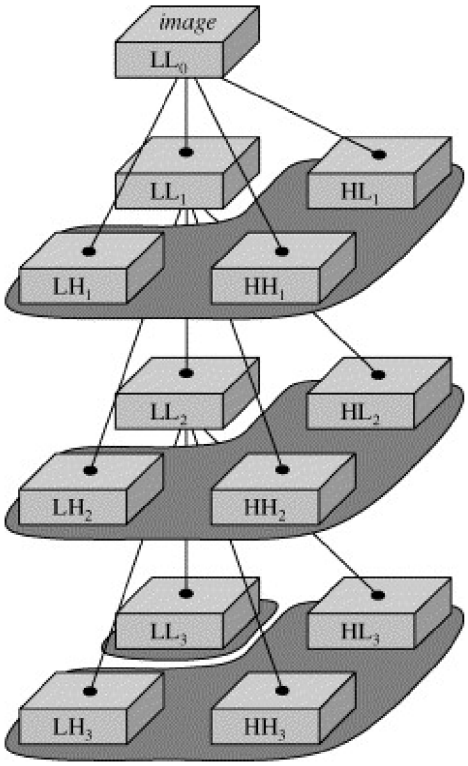
Inverzní ICT:

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1,0 & 0,0 & 1,402000 \\ 1,0 & -0,344136 & -0,714136 \\ 1,0 & 1,772000 & 0,0 \end{bmatrix} \begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} \tag{2.5}$$

2.2 DWT

Bavíme-li se o analýzu signálu pomocí funkce, tak vlnka znázorňuje sinusoidní nebo oscilační funkci v času nebo prostoru. Vlnková reprezentace slouží k analýze časově nezávislého signálu, kde nelze aplikovat statistické predikce. Typicky se jedná o diskontinuity u hran obrazu jak je uvedeno ve zdroji [9].

Diskrétní vlnková transformace umožňuje přenést obraz ve více rozlišení naráz. Na obrázku 2.2 lze vidět jak je zdrojový obraz LL_0 rozdělen na 4 podpásem značené jako LL_1, HL_1, LH_1 a HH_1. První písmeno značí typ frekvenčního lineárního filtru ve vertikálním zpracování, druhé písmeno značí typ frekvenčního lineárního filtru pro horizontální zpracování signálu. Typem může být horní propust nebo dolní propust. Následuje index podúrovně značící rozlišení obrazu.



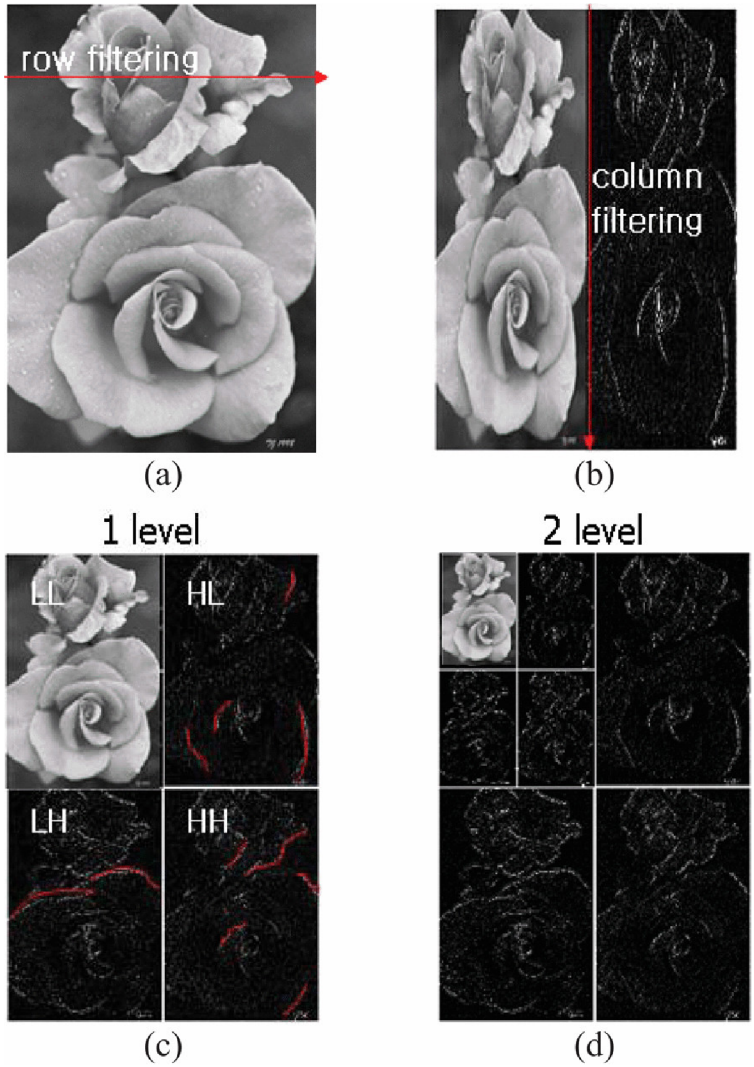
Obrázek 2.2: Rozdělení obrazu pomocí DWT na podúrovně LL, LH, HH, HL se čtyřmi úrovněmi rozlišení. Zdroj: [9]

Obraz je rozkládán do dalšího rozlišení hierarchicky do dalších 4 podpásem vždy z aktuálního podpásma LL. Proces rozkladu pokračuje až do požadované úrovně dané uživatelským vstupem. Celkový počet bodů ve všech podpásmech je identický k počtu bodů v rozkládaném obraze.

Vícerozměrné vlastnosti DWT vychází z faktu, že LL_d podpásmo je poloviční ve výšce a šířce než LL_{d-1} kde $d \geq 1$. Podpásmo LL_0 je podmásmo s nejvyšším rozlišením obrazu. Naopak nejmenší rozlišení je definováno jako LL_D kde $0 \leq d < D$. LL_d lze jednoduše obnovit z podpásem z rozlišení $d+1$ za pomocí úrovně D, a to aplikování syntézi $D-d$ úrovní DWT. To je výhodné, když chceme získat určité rozlišení obrazu.

Obrázek 2.3 ilustruje kroky transformace na fotografii. První je aplikováno filtrování po řádku, které vytvoří rozdělení na dvě podpásma dolní propust a horní propust (a) pomocí 1D vlnky. Následuje stejná akce, ale pro sloupcové zpracování (b) pro obě podmásma z minulého kroku. Výsledkem je rozdělení na 4 podpásma (c), kde jsou zvýrazněny magnitudy u signálu v řádkovém, sloupcovém a kombinací obou rozdělení. Varianta (d) již zobrazuje rozdělení do další úrovně rozlišení.

Dalším praktickým použitím DWT je získání bitové roviny ze všech podpásem daného rozlišení. U každého podpásma pixelu nás pak mohou zajímat jen nejvýznamnější bity. Zahozením méně významných bitů lze přirovnat ke kvantizaci originálního obrazu.



Obrázek 2.3: Příklad zpracování fotografie pomocí DWT. Zdroj: [11]

2.2.1 Dekompozice signálu

Pro dekompozici signálů je využita vícerozměrná analýza, která rozdělí signál na dvě části: aproximaci originálního signálu z jemného na hrubší rozlišení a detailní informace, které byly ztraceny při aproximaci. Matematická reprezentace je následující

$$f_m(t) = \sum_n a_{m+1,n}\phi_{m+1,n} + \sum_n c_{m+1,n}\psi_{m+1,n} \tag{2.6}$$

kde $f_m(t)$ označuje hodnotu vstupu funkce $f(t)$ v rozlišení 2^m. Část detailních informací je $c_{m+1,n}$ a část aproximace signálu $a_{m+1,n}$ pro rozlišení 2^{m+1}. Funkce $\phi_{m+1,n}$ a $\psi_{m+1,n}$ jsou dilatace a bázová vlnková funkce.

DWT využívá pro dekompozici signálů pyramidovou strukturu filtrů s konečnou pulzní odezvou (FIR). Je proto upravena předchozí rovnice pro výpočet vlnových koeficientů signálu $f(t)$.

$$
\begin{aligned}
c_{m,n}(f) &= \sum_k g_{3n-k} a_{m-1,k}(f) \\
a_{m,n}(f) &= \sum_k h_{2n-k} a_{m-1,k}(f)
\end{aligned} \tag{2.7}
$$

V rovnicích 2.7 je g filtr horní propusti a h filtr dolní propusti, $g_i = (-1)^i \; h_{-i+1}$ a $h_i = 2^{1/2} \int \phi(x-i)\phi_(2x)\,\mathrm{dx}$. Pro rekonstrukci signálu by šlo využít nekonečně mnoho filtrů h a g, nicméně z praktických důvodů je nutné mít konečný počet filtrů s konečnou impulzní odezvou. Propagací signálu přes soustavu filtrů je pak signál rekonstruován.

JPEG 2000 specifikuje v základu dva druhy vlnkových filtrů. První je biortogonální spline filtr (vlnky Cohen-Daubechies-Feauveau, CDF) 9/7 a druhá je vlnka 5/3. Analytický filtr CDF je tak složen z formace 9krokového filtru dolní propusti s konečnou impulzní odezvou a ze 7krokového filtru horní propusti s konečnou impulzní odezvou. Podle zdroje [1] jsou koeficienty pro dopřednou transformaci následující:

9kroková dolní propust: $[h_{-4}, h_{-3}, h_{-2}, h_{-1}, h_0, h_1, h_2, h_3, h_4]$

$$
\begin{aligned}
h_4 = h_{-4} &= +0,026748757410810 \\
h_3 = h_{-3} &= -0,016864118442875 \\
h_2 = h_{-2} &= -0,078223266528988 \\
h_1 = h_{-1} &= +0,266864118442872 \\
h_0 &= +0,602949018236358
\end{aligned}
$$

7kroková horní propust: $[g_{-3}, g_{-2}, g_{-1}, g_0, g_1, g_2, g_3]$

$$
\begin{aligned}
g_3 = g_{-3} &= +0,0912717631142495 \\
g_2 = g_{-2} &= -0,057543526228500 \\
g_1 = g_{-1} &= -0,591271763114247 \\
g_0 &= +1,115087052456994
\end{aligned}
$$

2.3 Kvantování

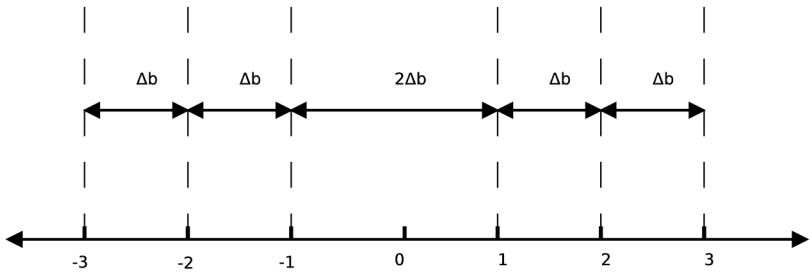
Po zpracování DWT jsou všechny podpásma kvantovány. Úkolem kvantování u ztrátové komprese je zmenšit přesnost podpásem, nebo-li zarovnat hodnoty na omezený počet úrovní, a tím docílit komprese. Kvantování je část enkodéru s největší úsporou datového toku. U bezeztrátové komprese se kvantizační krok neprovádí.

Kvantování má dvě varianty, skalární kvantizace a vektorová kvantizace. Ve skalární je každý vstupní symbol brán samostatně pro vytvoření výstupu. Jedná se o klasické sekvenční zpracování. Vektorová kvantizace potřebuje k vyprodukování výstupu skupiny ve formě vektoru. Shluk dat do jedné jednotky sice zmenší počet kvantizací, ale za cenu vyšší výpočetní náročnosti.

V první části standardu JPEG 2000 je kvantování definováno jako skalární kvantizátor s velikostí kroku Δ_b. Rozsah hodnot od střední zóny kvantizace je pak dáno $2\Delta_b$. Dolní index b značí identifikaci podpásma. Standard umožňuje různé velikosti kvantizačních kroků pro každé podpásmo. Krok může být dynamicky přizpůsoben podle rozsahu hodnot v každém podpásmu. Vztah pro uniformní skalární kvantizaci je následující:

$$q_b(i,j) = \text{sign}(y_b(i,j)) \left\lfloor \frac{|y_b(i,j)|}{\Delta_b} \right\rfloor \tag{2.8}$$

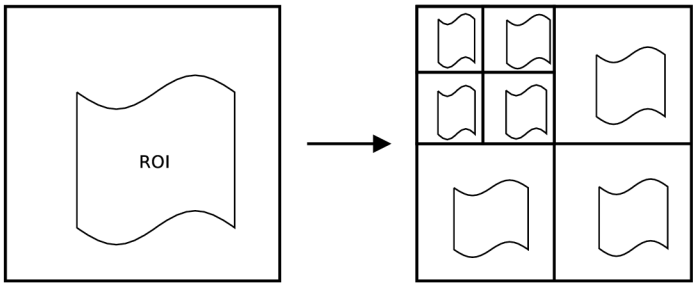
V rovnici 2.8 je $y_b(i,j)$ DWT koeficient podpásma b a Δ_b je velikost kvantizačního kroku pro podpásmo b. Všechny výsledné koeficienty $q_b(i,j)$ jsou celá čísla. Všechny výpočty v kvantizačním kroku jsou ve formě dvojkového komplementu. Při zvolení kroku 1.0 pak kvantizace není prováděna.



Obrázek 2.4: Ilustrace rozsahu kvantizačního kroku.

2.4 ROI

Jednou z unikátní vlastností formátu JPEG 2000 je funkce v originále nazvaná Region of interest (ROI). V následujícím textu je používán neoficiální český překlad *oblast zájmu*. Tato technika dovoluje pracovat s určitou ohraničenou částí obrazu, která může nabývat libovolného tvaru, jak je ukázáno na obrázku 2.5. Jednotlivé tvary se pak v průběhu zpracování rozloží na menší podčásti. Obraz je pak rozdělen na části důležité a části méně důležité. ROI je výhodné v situaci, kdy se potřebujeme zaměřit na určitou část obrazu. Např. chceme detail pouze obličejů lidí na obrázku a ostatní informace nás až tak nezajímají. Technika je využitelná u dalšího zpracování obrazu, např. u rozpoznávání obličejů hledaných osob nebo zaměření se na vysokofrekvenční změny a nízkofrekvenční jsou potlačeny.



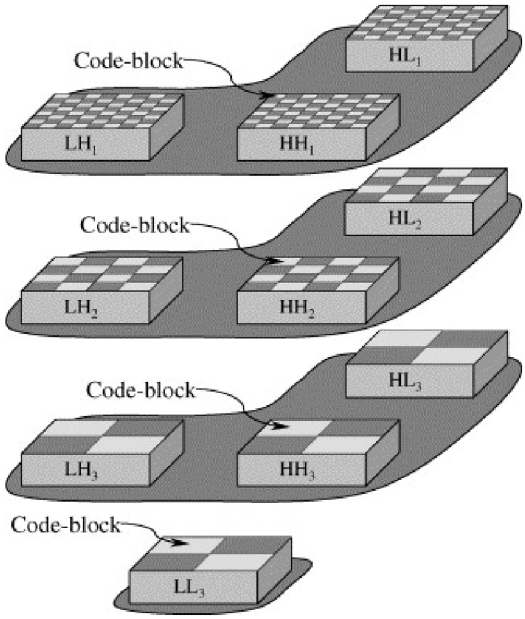
Obrázek 2.5: Příklad ROI oblasti aplikovaného na všechny podpásma obrazu.

2.5 EBCOT

Každé podpásmo je rozděleno na relativně malé části, např. o velikosti 64x64 nebo 32x32 pixelů, které nazýváme bloky (přeloženo z anglického výrazu *code-blocks*). Rozdělení ilustrováno v obrázku 2.5. Bloky obsahují výstup z DWT 2.2 v podobě vlnkových koeficientů.

Každý blok je zpracováván nezávisle a převáděn na bitový výstup c_i. Pro rozdělení do bloků je nutné si stanovit konečnou množinu bodů $Z_i + 1$ pro blok B_i. Délka konečné množiny bodů je stanovena jako $L_i^{(z)}$, kde

$$0 = L_i^{(0)} \leq L_i^{(1)} \leq \ldots \leq L_i^{(Z_i)}$$



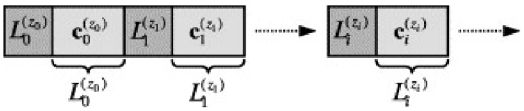
Obrázek 2.6: Rozklad podpásem na bloky. Každý blok má v tomto případě stejnou velikost napříč podpásmy. Zdroj: [9]

Při rekonstrukci obrazu se můžeme setkat s nepřesnostmi, které označíme jako sumu všech nepřesností z jednotlivých bloků. Dílčí nepřesnosti z bloku B_i označíme jako $D_i^{(Z)}$. Pokud je bitový výstup zarovnán na délku L_{max}, je vhodné si zvolit množinu bodů pro rozdělení takovou, aby byla celková suma nepřesností co nejmenší. Výběr rozdělovacích bodů je možné odložit až do doby, kdy bude znát počet rozdělovacích bodů z komprese bloků. Takové optimalizace se nazývá rate-disortion, která je probírána ve zdroji [8].

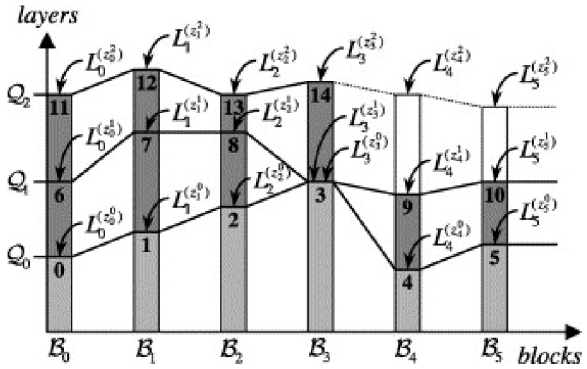
2.5.1 Vrstvy kvality

Výsledný datový proud formátu se skládá ze zabalených bitových proudů bloků. Samotný proud dat je ilustrován na obrázku 2.7. Optimálně zarovnány bitové výstupy bloků jsou konkatenovány, kde každý bitový výstup c_i je zaopatřen i o délku L_i. Při zájmu zpracovat jen danou oblast (ROI), lze jednoduše identifikovat pakety z celkového datového proudu, protože každá úroveň rozlišení obsahuje konečný počet bloků.

Pro získání menšího datového objemu a optimalizaci bloků nemáme zatím žádné prostředky pro laditelnou škálovatelnost. Pro vyřešení tohoto problému EBCOT zavádí vrstvy kvality. Na obrázku 2.8 je zobrazeno 6 po sobě následujících bloků. Každý blok může být rozdělen na α vrstev kvalit, kdy musí alespoň být v první vrstvě kvality. Výstupní proud dat se skládá z požadové úrovně kvality u každého bloku. Vrstva má pouze poznačené, kolik paketů se má zařadit z daného bloku do výstupního datového proudu.



Obrázek 2.7: Pakety opatřené o délku bitového proudu. Konkatenací vzniká datový proud. Zdroj: [9]



Obrázek 2.8: Vrstvy kvality v JPEG 2000. Čísla ve sloupci určují počet paketů potřebných k vytvoření datového proudu pro danou kvalitu. Zdroj: [9]

2.5.2 Výhody EBCOT

EBCOT paradigma nabízí několik výhod, v následujícím textu budou krátce shrnuty.

Flexibilní organizace: u vytváření datového proudu lze využít možností škálovatelnosti rozkladu na rozlišení, možnosti zkreslení (pokud jsou použity vrstvy kvality) a možnosti zaměření se na určité části obrazu (ROI).

Multiprocesní zpracování: každý blok je zpracováván samostatně, lze tak využít možností paralelismu pro zpracování více bloků naráz.

Minimalizace chybovosti: Chyby objevené uvnitř bitového proudu bloku nebudou mít vliv na ostatní bloky. Chyba nebude tak kumulována od zdroje k okolním blokům.

2.6 Kódové algoritmy

Po DWT a kvantování následuje fáze enkódování, které jsou nazvány jako *Tier-1* a *Tier-2*. V této práci se zaměříme na kodér entropie *Tier-1*, který je kombinací kodéru bitových rovin a aritmeticko-binárním kodérem. Ve standardu JPEG [4] je použita jiná dvojice kodérů: Hufmanovo kódování a RLE.

V případě kodéru bitových rovin se snažíme minimalizovat symetrie a redundance uvnitř bitové roviny, ale i skrz více bitových rovin. EBCOT enkóduje každou bitovou rovinu ve třech fázích (SPP, MRP, CUP). Bitová rovina je rozdělena na části, kde figuruje roli pouze jedna fáze bez přesahu. Proto je kodér bitových rovin nazýván také jako **dílčí** kodér bitových rovin. Fáze v kodéru vyjmenujeme v pořadí, jakém jsou prováděny:

- **Significance Propagation Pass (SPP)**: Bitové pozice, které mají magnitudu poprvé rovnu 1 jsou kódovány v touto fází.

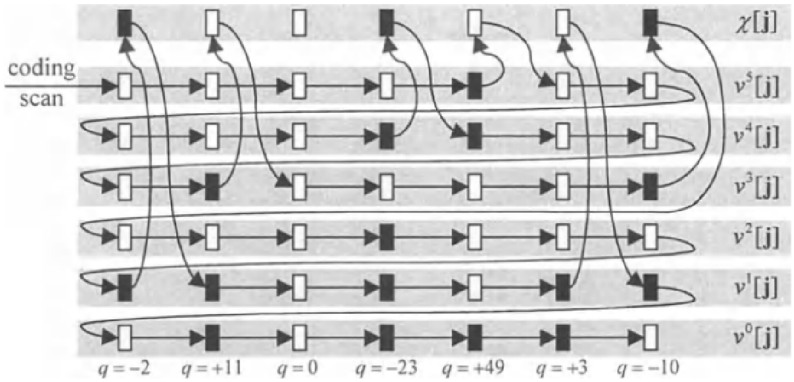
- **Magnitude Refinement Pass (MRP)**: Bitové pozice, které nebyly kódovány SPP a neměli magnitudu rovnu 1 v předchozích bitových rovinách (současný bit není nejvýznamnějším bitem současného koeficientu vzorku) jsou kódovány touto fázi.
- **Cleanup pass (CUP)**: Bitové pozice, které nebyly kódovány v předchozích průchodech jsou kódovány touto fázi. Také odstraňuje sekvenci nul na vstupu, jako v RLE.

2.6.1 Kodér bitových rovin

Existuje sekvence vzorků $y[j] \equiv y[j_1, y_2]$ podpásem patřící do relevantního bloku, které mají výšku J_1 a šířku J_2. Omezení indexů je pak $0 \leq j_1 < J_1$ a $0 \leq j_2 < J_2$. Pro blok j po kvantizaci má část znaménka $\chi[j]$ a magnitudy $\upsilon[j]$. Pole *magnitud* υ je vyplněno bezznaménkovými celými čísly. Hodnota na každém indexu je rovna absolutní hodnotě bloku na stejném indexu. Pole magnitud je tak stejně velké jako blok.

Magnitudy jsou rozděleny na *bitové roviny* $\upsilon^p[j]$. Bitová rovina znázorňuje pak jednu bitovou pozici napříč všemi magnitudami. Počet bitových rovin je omezen na K jednotek.

Jako první se zpracuje bitová rovina obsahující nejvíce významné bity $\upsilon^{K-1}[j]$ postupně po j bitech. Pokud se hodnota $\upsilon^{K-1}[j] \neq 0$, pak je do streamu propagováno znaménko $\chi[j]$. V další bitové rovině $\upsilon^{K-2}[j]$ se postupuje obdobně. Propagace znaménka na výstup je však podmíněna, pokud v předchozích bitových rovinách na stejné pozici j nebylo znaménko v minulosti propagováno. Proces pokračuje v dalších magnitudách dalšího bloku. Celou situaci ilustruje obrázek 2.9.



Obrázek 2.9: Proces kodéru bitových rovin. Nenulové bity magnitud a záporné bity znamének jsou začerněny v obdélníčkách. Zdroj: [10]

2.6.1.1 Skenovací vzor

Pro kódování je nutné určit, jakou sekvenci bloků budeme zpracovávat. Zavedeme tedy pojem *skenovací vzor*. Bloky jsou konceptuálně rozděleny do proužků (angl. *stripe*), které jsou vysoké 4 řádky bloků. V případě že obraz nemá počet bloků ve sloupci dělitelným čtyřmi, je u posledního proužku vzata zbývající výška. Průchod bloků v proužku nazýváme skenovací vzor. Standardní skenovací vzor prochází bloky vertikálně, po překročení výšky proužku pokračuje v dalším sloupci.

JPEG 2000 definuje dva přístupy u průchodu pomocí proužků. Jedním je regulérní a druhý vertikální přirozený. Liší se mezi sebou tím, že vertikální přirozený nebere vpotaz

při enkódování sousední bloky z předchozího a následující proužku v prvním a poslední řádku aktuálního proužku. Nejsou tedy propagovány do okolních proužků žádné informace z okolí. Tuto situaci lze přirovnat k zpracování proužků u okraje obrazu.

Vertikální skenovací vzor však trpí neduhy v práci s pamětí. Moderní procesory totiž při načtení jednoho bloku v matici přednačítají do vyrovnávací paměti i další bloky, které jsou uloženy sekvenčně po aktuálním bloku v řádku. Ve vertikálním posunu tak dochází k častému výpadku paměti a nutnosti přednačíst do paměti nové bloky, které se vůbec nepoužijí. V kapitole řešení je toto téma ještě rozebíráno a jsou nabízeny možnosti k řešení tohoto problému.

2.6.1.2 Stavové proměnné

Při kódování bloků jsou vytvořeny 3 stavové pole δ, $\delta^{'}$ a η o stejné velikosti jako blok. Zpočátku jsou všechny pole inicializovány na hodnotu 0. Stavové proměnné $\delta[m,n]$ a $\eta[m,n]$ se mohou v průběhu kódování změnit na hodnotu 1, nikdy se už ale nevrátí zpět na hodnotu 0, dokud celý blok není zpracován. Interpretace hodnot je následující.

- Když je $\delta[m,n] = 1$, znamená to, že první nenulový bit $\upsilon[m,n]$ v řádku m a sloupci n byl kódován. Jinak je roven 0.
- Když je $\delta^{'}[m,n] = 1$, označuje to, že operace *magnitude refinement coding* byla aplikována na $\upsilon[m,n]$. Jinak je roven 0.
- Když je $\eta[m,n] = 1$, znamená to, že operace *zero coding* byla aplikována na $\upsilon^{P}[m,n]$ v *significant propagation pass*. Jinak je roven 0.

2.6.1.3 Zero coding tabulky

Operace *zero coding* potřebuje ke kódování 3 převodní tabulky 2.2, 2.3 a 2.4. Písmeno x v řádcích tabulek znamená libovolná hodnota. Převodní tabulky se opírají o hodnoty stavové proměnné δ z osmiokolí elementu a vyprodukují kontextovou značku CX. Okolí je pak znázorněno v tabulce 2.1. Element, který nás zajímá označíme X. V okolí si označíme písmenem H horizontální sousedy, písmenem V vertikální sousedy a písmenem D sousedy v diagonále.

D_0	V_0	D_1
H_0	X	H_1
D_3	V_1	D_2

Tabulka 2.1: Okolí elementu X pro *zero coding*.

2.6.1.4 Operace pro kódování

V EBCOT jsou definovány 4 kódové operace pro vygenerování kontextu CX a rozhodovacího bitu D. Tyto dvě informace potřebuje aritmetický kodér 2.6.2 pro vygenerování bitového proudu. Kontext je bezznaménkové celé číslo $CX \in (0-18)$, zatímco $D \in \{0,1\}$. Index aktuální bitové roviny označíme písmenem P. Nyní si představíme kódové operace:

- **Zero coding (ZC)**: rozhodovací bit $D = \upsilon^{P}[m,n]$ a kontextová značka CX je vybrána z jedné převodní tabulky pro *zero coding* podle podpásma (LL, LH, HL nebo

Podpásma LL a LH			**Kontextová značka**
$\sum H$	$\sum V$	$\sum D$	CX
2	x	x	8
1	≥ 1	x	7
1	0	≥ 1	6
1	0	0	5
0	2	x	4
0	1	x	3
0	0	≥ 2	2
0	0	1	1
0	0	0	0

Tabulka 2.2: Tabulka pro *zero coding* u bloku z podpásma LL a LH.

Podpásmo HL			**Kontextová značka**
$\sum H$	$\sum V$	$\sum D$	CX
x	2	x	8
≥ 1	1	x	7
0	1	≥ 1	6
0	1	0	5
2	0	x	4
1	0	x	3
0	0	≥ 2	2
0	0	1	1
0	0	0	0

Tabulka 2.3: Tabulka pro *zero coding* u bloku z podpásma HL.

Podpásmo HH		**Kontextová značka**
$\sum(H+V)$	$\sum D$	CX
x	≥ 3	8
≥ 1	2	7
0	2	6
≥ 2	1	5
1	1	4
0	1	3
≥ 2	0	2
1	0	1
0	0	0

Tabulka 2.4: Tabulka pro *zero coding* u bloku z podpásma HH.

HH) aktuálního bloku. Výběr příslušného řádku je probírán v podsekci *Zero coding tabulky*.

- **Sign coding (SC)**: D a CX je určen podle horizontální referenční hodnoty H a vertikální referenční hodnoty V, které získáme pro lokalitu $[m, n]$ následujícím způ-

sobem.

$$H = \min[1, \max(-1, \delta[m, n-1] \times (1 - 2\chi[m, n-1]) + \delta[m, n+1] \times (1 - 2\chi[m, n+1]))]$$

$$V = \min[1, \max(-1, \delta[m-1, n] \times (1 - 2\chi[m-1, n]) + \delta[m+1, n] \times (1 - 2\chi[m+1, n]))]$$

Referenční hodnoty H a V pak nabývají tří možných stavů. Důležitost nám pak říká, že stavová proměnná δ je rovna 1, nebo při nedůležitosti je rovna 0.

1. **0** indikuje oba sousedy jako nedůležité, nebo důležité, ale mají opačná znaménka.
2. **1** indikuje, že jeden nebo oba ze sousedů má kladné znaménko a je důležitý.
3. **−1** indikuje, že jeden nebo oba ze sousedů má záporné znaménko a je důležitý.

Referenční hodnoty H a V pak určí podle tabulky 2.5 kontext DX. Rozhodovací bit $D = \hat{\chi} \oplus \chi[m, n]$, kde $\oplus$ reprezentuje binární operaci výlučné logické nebo.

H	V	$\hat{\chi}$	CX
+1	+1	0	13
+1	0	0	12
+1	−1	0	11
0	+1	0	10
0	0	0	9
0	−1	+1	10
−1	+1	+1	11
−1	0	+1	12
−1	−1	+1	13

Tabulka 2.5: Referenční tabulka pro *sign coding*.

- **Magnitude refinement coding (MRC)**: rozhodovací bit D na pozici (m, n) v P bitové rovině je roven hodnotě $v^P[m, n]$. Kontext CX závisí na $\delta'[m, n]$ a sumě osmiokolí stavové proměnné δ:
  - Pokud $\delta' = 1$ na aktuální pozici, což poukazuje, že se nejedná o první magnitude refinement pro tento element, pak $CX = 16$
  - Pokud $\delta' = 0$ na aktuální pozici a suma hodnot δ v osmiokolí je rovna 0, pak $CX = 14$
  - Pokud $\delta' = 0$ na aktuální pozici a suma hodnot δ v osmiokolí je větší než 0, pak $CX = 15$
- **Run-length coding (RLC)**: Operace počítá s maximálně 4 sousedními bity (vzorek) v skenovacím vzoru proužku. Kódování může vygenerovat jeden rozhodovací bit nebo rovnou 3, podle toho, jaký je počet jedniček v proužku. $CX = 17$ při naskytnutí se první jedničky, nebo pokud jsou ve vzorku samé 0. Po nalezení první jedničky je $CX = 18$. Rozhodovací bit D je roven 0, pokud jsou všechny bity rovny 0, jinak je $D = 1$.

2.6.1.5 Fáze kodéru

Na každou bitovou rovinu bloku jsou aplikovány 3 fáze kodéru v pořadí Significance Propagation Pass, Magnitude Refinement Pass a Cleanup pass. Vyjímkou se stává první bitová rovina, kde je aplikován pouze Cleanup pass. Po dokončení každé kódovací fáze ve skenovacím vzoru, další kódovací fáze restartuje skenovací vzor na počáteční pozici.

- **Cleanup pass (CUP)**: Je aplikován na každou bitovou rovinu bloku, co je dokončen MRP, kromě první bitové roviny, kde MRP není potřeba. V každé pozici (m, n) CUP zkontroluje $\delta[m, n]$ a $\eta[m, n]$, jestli nejsou oba rovny 0. Pokud žádný není hodnoty 0, pak je proveden posun na další bitovou pozici v bitové rovině. Pokud jsou oba rovny 0, pak je spuštěn run-length coding (RLC) nebo zero coding (ZC). RLC je spuštěn jen za splnění následujících předpokladů:

  1. index m je násobkem 4, včetně $m = 0$
  2. $\delta = 0$ v celém sloupci proužku o výšce 4 bloků
  3. $\delta = 0$ pro všechny přidružené sousedy bloků v celém sloupci proužku

  Pokud je alespoň jeden předpoklad nepravda, pak je spuštěno ZC. Po dokončení RLC nebo ZC zkontrolujeme před posunutím na další bit, pokud je nutné provést sign coding (SC). Pokud v poslední kódové pozici $\upsilon^P[m, n] = 1$, znamená to, že aktuální bit je nejvýznamnější bit, pak $\delta[m, n]$ je nastaveno na hodnotu 1 hned po ZC nebo RLC. Kódování pokračuje, dokud nejsou všechny bity ve skenovacím vzoru bitové roviny kódovány. Po dokončení CUP je resetována stavová proměnná $\eta[m, n] = 0$ pro celou bitovou rovinu, před přesunutím na další bitovou rovinu.

- **Significance Propagation Pass (SPP)**: SPP aplikuje jako první zero coding, pokud alespoň jeden z osmi sousedů má $\delta = 1$ a aktuální pozice $\delta[m, n] = 0$. Pokud zero coding nelze provést, pak se přejde na následující bitovou pozici. V opačném případě se nastavuje $\eta[m, n] = 1$. Po zero coding se provede kontrola, pokud na aktuální pozici je $\upsilon^P[m, n] = 1$, pak se provede sign coding a nastavíme $\delta[m, n] = 1$.

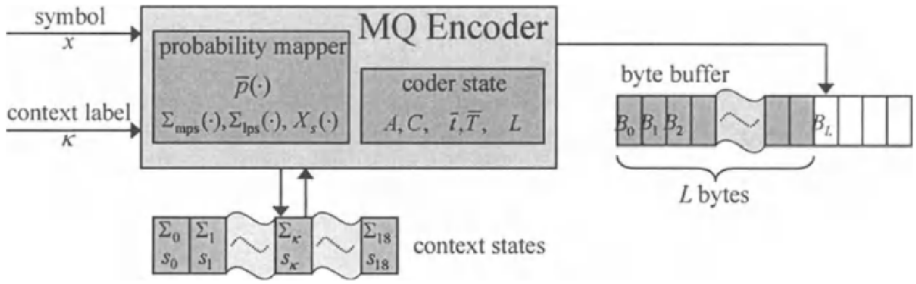
- **Magnitude Refinement Pass (MRP)**: Pokud stavové proměnné $\delta[m, n] = 1$ a $\eta[m, n] = 0$, pak aplikujeme MRC a nastavíme $\delta'[m, n] = 1$. Pokračuje kódování na dalších bitových rovinách.

2.6.2 Aritmetický kodér MQ

Kodér bitových rovin počítá s předzpracováním binárních symbolů, aby mohl efektivně převést symboly na datový proud. Předzpracování se v tomto případě nazývá aritmetický kodér, známý jako MQ kodér. Tato sekce je převzata ze zdroje [10].

MQ kodér si lze představit jako stroj, který mapuje sekvence symbolů $x_n \in \{0, 1\}$ a přidružené kontextové značky k_n v jedno kódové slovo. Komprimované kódové slovo reprezentuje jen část většího komprimovaného bitového výstupu. Tuto část nazveme jako segment. Segment je postupně doplňován novými poznatky, jakmile je na vstupu dvojice symbol a kontext (x_n, k_n). Celá situace je ilustrována v obrázku 2.10. Algoritmus pro doplňování poznatků se skládá ze 3 částí:

1. Jsou stanoveny stavové proměnné A, C, t, TaL. Pro identifikaci stavu v určité fázi, pro určitý vstup, zavedeme notaci proměnné s dolním indexem n. A a C identifikují délku



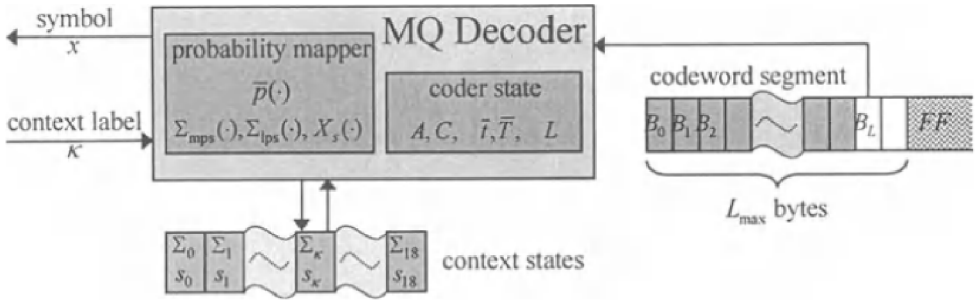
Obrázek 2.10: Schéma MQ kodéru. Zdroj: [10]

intervalu a kruhový posuvný registr. Proměnná L reprezentuje počet již vygenerových kódových bytů. T je dočasná vyrovnávací paměť. t je sestupný čítač, který při hodnotě $t = 0$ spustí akci přesunutí generovaného kódu z registru C do dočasné vyrovnávací paměti T.

2. Kontextový stavový registr obsahuje párové záznamy (Σ_k, s_k) pro každou kontextovou značku k. Bit $s_k \in \{0, 1\}$ označuje nejvýznamnější symbol pro kódový kontext s indexem k. Σ_k je 6bitové číslo znázorňující pravděpodobnost (rozsahu 0 až 46) nejvýznamnějšího kódového kontextu.

3. Kódový kontext (Σ_k, s_k) je mapován do 4 pravidel pomocí funkcí (vyhledávacích tabulek). Tabulku lze nalézt v knize [10]. Jedná se o mapování mezi stavovou proměnnou Σ_k a pravděpodobnosti méně častého symbolu (LPS) pro kontext k. Nová hodnota Σ_k pro další stav je určena podle toho, jestli aktuální kódovaný symbol je nejvíce pravděpodobný symbol (MPS) ($x_n = s_{k_n}$) nebo LPS ($x_n = 1 - s_{k_n}$). Po vyprodukování LPS symbolu je MPS a LPS prohozeno, pokud ne, pak s_k je nahrazeno $1 - s_k$.

Pro pochopení kódování popíšeme ještě ve stručnosti MQ dekodér. Dekodér na vstupu získá sekvenci kontextových značek k_n z dekodéru bitových rovin a na výstup přivede dekódované symboly $x_n \in \{0, 1\}$. Dekodér zpracovává byty z komprimovaného segmentu kódového slova. Jak je vidět na obrázku 2.10 a 2.11, struktura kodéru a dekodéru se ve výstavbě architektury moc neliší.



Obrázek 2.11: Schéma MQ dekodéru. Zdroj: [10]

Důležitou poznámku je, že segment dostupný pro dekodér není většinou identický jako u kodéru. Když dekodér se dostane za validní část datového segmentu, hodnoty které bude

dekodér mít na vstupu budou rovny speciální hodnotě FF_h (255 v dekadickém zápise). Kodér zarovnává počet bytů v segmentu pomocí těchto speciálních hodnot FF_h, kdy poslední byte segmentu je vždy roven FF_h.

Problém nastává kdy na výstup kodéru se má propagovat hodnota FF_h, je nutné tuto hodnotu normalizovat do rozsahu od 00_h do $8F_h$. Ostatní hodnoty v rozsahu od 90_h FF_h jsou použity jako značky (nebo výplň, jak je uvedeno v minulém odstavci) ve výstupním datovém proudu. Příkladem může být značka začátek dlaždice *SOT* (start of tile-part) nebo začátek paketu *SOP* (start of packet).

2.7 Shrnutí

V kapitole č. 2 jsme si ukázali, jaké podsystémy zpracovávají vstupní signál až do výstupního datového proudu. Formát JPEG 2000 je koncipován do velké volitelnosti, kdy lze použít při zpracování jen nějaké podčásti. To nám dovoluje použít jen určité části a najít východisko mezi výkonem a mírou komprese. Lze tak cílit na určitou podmnožinu problémů, které cheme řešit v reálném světě. U CT snímků se může jednat o možnosti více vrstev a selektivní enkódování. Pochopení formátu je tak prerekvizitou k samotné implementaci, která je probírána v následující kapitole.

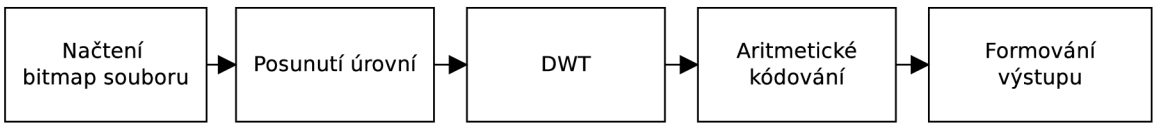
Kapitola 3

Řešení

Pro porovnání bude v této kapitole srováváno vlastní řešení se vzorovou implementací standardu OpenJPEG a implementací FFmpeg. Všechny tři řešení jsou implementovány v jazyce C. Jazyk C byl vybrán z důvodu svobodné manipulace s různými datovými typy, možnosti modulární struktury kódu a samostatné správě využití paměti.

Na obrázku 3.1 jsou ilustrovány kroky v kódovacím procesu. Jako první je provedeno načtení vzorových dat do společných datových struktur z formátu BMP. Vstupní data jsou pak rozdělena na jednotlivé obrazové komponenty, které jsou zpracovávány nezávisle. Jako první se provede předzpracování v podobě posunutí úrovní a převodu barevného modelu. Následně je aplikována dekompozice do rozlišení, podpásem a bloků pomocí DWT. Zpracování pokračuje aplikováním aritmetického kodéru na jednotlivé bloky a výstupem je již datový segment obsahující komprimovaná data.

Na konci kapitoly je uvedeno měření tří různých implementací, jaké nástroje jsou k tomu použity a na jakých datech se testuje. Výstupní testovací data jsou srovnány v měřítku velikosti rozlišení vůči celkovému času enkódování na jeden pixel.



Obrázek 3.1: Blokové schéma implementace.

3.1 Předzpracování

3.1.1 Načtení vstupního obrazu

V implementaci načtení vstupního obrazu je použit obrazový formát BMP (BitMaP). BMP byl navržen jako základní rastrový obrazový formát pro operační systémů firem IBM a Microsoft. Formát byl vybrán pro jeho vlastnosti jednoduché manipulace bez použití externí knihovny a podporu v rastrových editorech. Formát BMP obsahuje dvě hlavičky, a to hlavičku souboru a informační hlavičku o obrázku. V informační hlavičce jsou obsaženy informace o velikosti obrázku (šířka a výška), počet barevných komponent, počet bitů na komponentu, typ komprese, a další informace. V hlavičce souboru nás zajímá paměťový posun v bytech od začátku obsahu souboru k prvnímu pixelu obrazu. Ve zdroji [7] lze na-

lézt plnou specifikaci BMP formátu. Jednotlivé pixely obrazu jsou za sebou poskládány sekvenčně jako 2D matice bodů.

Při zpracování se postupuje od levého horního rohu jako pozice (x, y), kde $x = 0, y = 0$ a pokračuje se inkrementací po ose x k šířce obrazu. Po překročení šířky je zpracováván další řádek v ose y. Proces se opakuje až po vyčerpání všech bodů. Program při načtení podporuje pouze BMP TrueColor formát o 24 bitech na jeden pixel, kdy každá barevná složka má 1B. Pixel má strukturu tří barvových složek v pořadí BGR, tedy 8 bitů pro modrou složku, 8 bitů pro zelenou složku a 8 bitů pro červenou složku. Pro přečtení z bitmapy je nutné počítat se zarovnáním na 32bitů. K přečtení všech dat je určena velikost kroku *stride* mezi jednotlivými řádky obrazu. Velikost řádku je dána *width* a *bitCount* je počet bitů na pixel (v našem případě 24).

```
stride = ((width \times bitCount + 31) / 32) \times 4);
```

K výpočtu adresy jednoho pixelu je použit následující útržek kódu, kde *height* je výška obrazu:

```
pixelAddress = (height - 1U) \times stride;
```

Jednotlivé složky jsou pak v paměti uloženy separátně, a proces kódování je tak aplikován po jednotlivých složkách. Tato implementace načítá vstupní data pomocí standardní vstupní knihovny a její funkce *fread*, která načítá po proužcích data do alokovaného prostoru o velikost získané z hlavičky BMP souboru.

3.1.2 Transformace zdrojových dat

Nad jednotlivými složkami je provedena normalizace dat pro pozdější proces kódování podle standardu JPEG 2000. Jako první je proveden posun úrovně barevné složky. Z bitmap formátu byla načtena barevná informace o velikosti 8 bitů s rozsahem hodnot v intervalu (0, 255). Jak bylo probíráno v druhé kapitole v sekci 2.1.2, interval hodnot je posunut na interval se středem kolem 0. Jednoduše se odečte od zdrojové hodnoty hodnota $2^7 - 1 = 127$.

Po posunu je aplikován převod na YUV formát, který odděluje informaci o jasu barvy a barevnonosných složek. Je proto provedena dopředná ICT transformace, kdy na RGB složky pixelu je aplikována konverzní matice. Rovnice pro převod je uvedena 2.4. Výsledkem je složka o jasu barvy (Y), a barvonosné informace (Cb, Cr). Pro úsporu času jsou obě operace (posunutí a převod barevného formátu) provedeny naráz v jednom průchodu pixely.

3.2 DWT

Tato sekci čerpá z [3] a pojednává o existujících metodách počítání 2D diskrétní vlnkové transformace. V počítačích dnešní doby je dostupná vícevrstvá vyrovnávací paměť procesoru. Při návrhu DWT je dobré využít vlastností vyrovnávací paměti, které pracují typicky s paměťovými proužky o velikosti 64 bytů. Proto je vhodné informace o signálu rozčlenit na menší kusy namísto načtení celého signálu.

DWT rozčlení signál pomocí filtrů na horní propust a dolní propust. U 1D transformace je využita dekompozice těchto filtrů do takzvaného pyramidového schéma. Naivní algoritmus pro 2D transformaci by pak provolával sérii 1D transformací horizontálně a pak další sérií pro vertikální zpracování. Lepší situace je použita v implementaci OpenJPEG, která využívá tzv. *lifting schema*, které využívá predikce a aktualizace na stejném paměťovém místě. Nezaručí ale, že se při horizontálním a vertikálním zpracování objeví přesahy v čtení

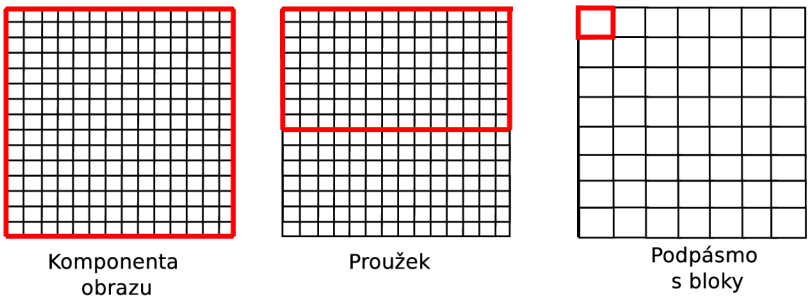
paměťového prostoru. V obou případech, jak u pyramidového schéma, tak u lifting schema jsou vlnkové koeficienty navštíveny alespoň dvakrát, jednou při horizontálním a podruhé při vertikálním průchodu. Proces je tak zatěžován přístupem do mezivýsledků transformace. Navíc, při zpracování koeficientů ve vertikálním režimu, dochází při každém kroku k výpadku vyrovnávací paměti.

Pro DWT je využita knihovna od pana Ing. Davida Bařiny, Ph.D., která využívá zpracování dat po proužcích, takže jsou data dostupná ve vyrovnávací paměti během jednoho kroku transformace. Vstupní data jsou transformována kontinuálně v pruzích o výšce 2×2^{c_h}, kde c_h je rovno výšce jednoho výstupního bloku. Proužková transformace je provedena v jednoprůchodovém režimu. Pro generování bloků je potřeba celkem $(2S + 3 \times 2^{c_h})M$ vzorků v paměti, kde M je šířka obrazu, S je počet kroků *lifting* procesu. U proužkového zpracování jsou použity dva pomocné odkládací prostory v paměti. Jeden slouží pro odkládání dat v horizontálním směru a druhý ve vertikálním směru. Oba dva mají velikost rovnu 4 blokům. Tato paměť je znovu používána pro zpracování všech podpásem v celé transformaci. Při dalším proužku je vertikální buffer předán dalšímu bloku k dalšímu zpracování. Horizontalní buffer bude použit pro další proužek ležící pod aktuálně zpracovávaným proužkem. Navíc jsou alokovány dvě paměťová místa M_j a N_j pro každé podpásmo, kde j je úroveň zanoření podpásma. Velikost bufferů je pro počáteční a také největší podpásmo stejné, jako počet bloků. S úrovní dekompozice se velikost bufferů půlí (se zaokrouhlením nahoru). Všechny bloky jsou tedy vyprodukovány v jednom průchodu a nadále se pracuje pouze s výměnou offsetu pro získání daného proužku.

Na obrázku 3.2 lze vypozorovat, jak jednotlivé komponenty obrazu jsou transformovány do bloků. Postup je tedy následující. Zpracovávána je jedna komponenta obrazu za druhou, nad kterou je provoláváno rozdělení na proužky, které se vzájemně nepřesahují. Při rozdělení obrazu na jeden proužek je zavolána transformace, která aplikuje vlnku CDF 9/7. Pro více rozlišení je dekompozice volána rekurzivně. V tabulce 3.1 je reprezentována struktura výstupu použité knihovny.

Typ proměnné	**Význam**
pole float koeficientů	separované LL podpásmo rozlišení
pole float koeficientů	separované HL podpásmo rozlišení
pole float koeficientů	separované LH podpásmo rozlišení
pole float koeficientů	separované HH podpásmo rozlišení
dvojice (w,h)	velikost bloku v každém podpásmu (šířka, výška)
celé kladné číslo	úroveň rozkladu rozlišení
dvojice (x,y)	souřadnice začátku (včetně) validních dat
dvojice (x,y)	souřadnice konce (mimo) validních dat
dvojice (w,h)	velikost validních dat (šířka, výška)
celé kladné číslo	index rozlišení
celé kladné číslo	lineární sekvenční číslo bloku skrz všechny podpásma
celé kladné číslo	lineární sekvenční číslo bloku v LL podpásmu
celé kladné číslo	lineární sekvenční číslo bloku v HL podpásmu
celé kladné číslo	lineární sekvenční číslo bloku v LH podpásmu
celé kladné číslo	lineární sekvenční číslo bloku v HH podpásmu

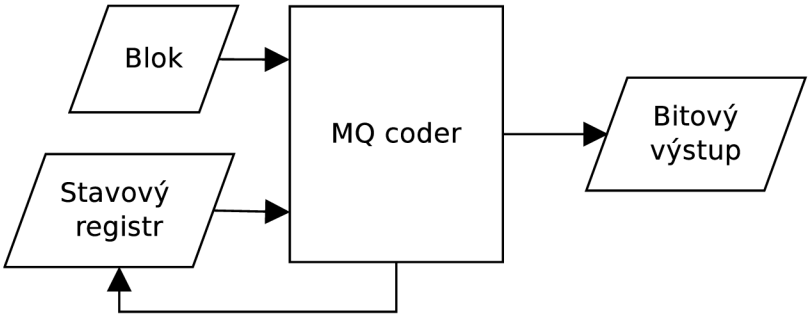
Tabulka 3.1: Struktura bloku po DWT transformaci.



Obrázek 3.2: Zjednodušené schéma transformace obrazu.

3.3 Aritmetický kodér

Při DWT jsou transformovány jednotlivé komponenty do bloků, které jsou zpracovávány postupně, podle velikosti proužku. Nyní tedy pracujeme s jednotlivými bloky dat určitého rozlišení. V implementaci byl převzat aritmetický MQ kodér ze zdrojového kódu knihovny OpenJPEGm který na vstupu očekává symboly ve formátu 32bitového čísla. DWT ale vrací na výstup vlnkové koeficinety ve formě čísla s pohyblivou desetinou čárkou. Proto je před kódováním provedena konverze. Na obrázku 3.3 lze vidět, jak je na vstup přiveden blok a stavový registr, který je v průběhu komprese aktualizován. Na výstupu je přiveden bitový proud. V druhé kapitole v sekci 2.6.2 je dopodrobna probírám způsob kódování.



Obrázek 3.3: Ilsutrace aritmetického kodéru z pohledu vstupů a výstupů.

3.4 Výstupní formát

Jednotlivá data z bloků jsou přivedeny do výstupního streamu ve formě segmentů. Nicméně celý formát není založen pouze na obrazových datech, ale i o hlavičkách, které dynamicky určují kódovací parametry. Jedním takovým příkladem je ROI, tedy oblasti s možnostmi měnit kvalitu obrazu, ale i další parametry. Pro oddělení typu segmentu a pro jednoduchou manipulaci při dekódovacím procesu jsou použity v datovém streamu speciální značky. Standard jasně vymezuje hodnoty pro značky, které vždy začinají bytem s hodnotou FF_h.

Pro základní implementaci byl vytvořen zjednodušený formát JPEG 2000, kde jsou použity jen některé značky, a to začátek obsahu (SOC), velikost obsahu (SIZ), začátek paketu (SOP) a konec obsahu (EOC). Celý výstupní formát je vyobrazen tabulkou 3.2. Ve formátu se opakuje sekvenčně pouze část s paketem, kde jsou uschována komprimovaná data. Při dekompresi je značka EOC brána jako zarážka, program pak nemusí pokračovat ve

zpracování dalšího paketu. Implementovaný formát je vcelku uzavřený na možnosti JPEG 2000.

Zkratka	**Velikost**	**Definovaná hodnota**	**Význam**
SOC	2B	$FF4F_h$	začátek obsahu
SIZ	2B	$FF51_h$	začátek segmentu s velikostmi
imageWidth	4B		šířka obrazu
imageHeight	4B		výška obrazu
imageOffsetX	4B		posun začátku obrazu v ose x
imageOffsetY	4B		posun začátku obrazu v ose y
resolutions	4B		počet úrovní rozlišení
comps	4B		počet komponent obrazu
cblkWidth	4B		šířka bloku
cblkHeight	4B		výška bloku
SOP	2B	$FF91_h$	začátek paketu
compNo	4B		index komponenty
resolutionNo	4B		index rozlišení v komponentě
bandNo	4B		index podpásma
cblkNo	4B		sekvenční index bloku
packetLength	4B		délka paketu
packetBytes	xB		data paketu dána délkou
EOC	2B	$FFD9_h$	data paketu dána délkou

Tabulka 3.2: Reprezentace struktury formátu.

3.5 Měření

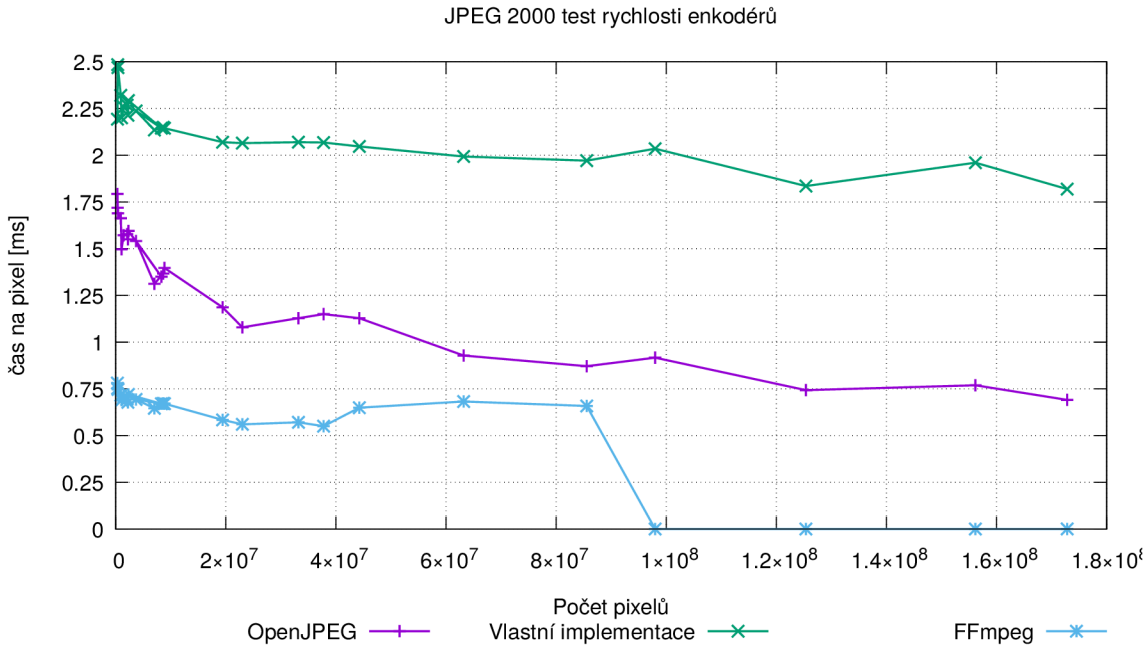
Pro testování byl vytvořen pomocný skript, který postupně testoval na předchystaných obrázcích různého rozlišení výkon a paměťovou náročnost implementace. Výsledky je možné vidět na obrázku 3.5 a v obrázku 3.6. Testování proběhlo nad otevřenou knihovnou OpenJPEG, která je implementována v jazyce C, tak jako implementace vlastního řešení.

Testování bylo provedeno na počítači s procesorem Intel Core i5-3210M, který umožňuje paralelní zpracování ve 4 vláknech. Po paměťové stránce je testovací stroj vybaven 8 GB RAM. Velikost paměti byla také limitem u implementace FFmpeg, kdy při nejvyšších rozlišeních obrázku počítač situaci nezvládal vyčerpáním paměti. V měření jsou tedy tato místa ponechána jako nevyplněná pro u sloupce pro FFmpeg.

Při testování výkonu byla měřena časová jednotka na jeden pixel, která byla sledována nad počtem 25 rozlišeních. Jedná se o rozlišení často používaná u monitorů a displejů počítačů, v ditálních formátech televizního vysílání a digitální projekce kina. V poslední řadě bylo provedeno měření na obrázku až do rozlišením s více jak 100 megapixelů. Celý přehled použitých rozlišení je možné nalézt v tabulce 3.3.

U paměťové náročnosti je testován maximální alokovaný paměťový blok při enkódovacím procesu. Na měření je použit program *time*, který je dostupný ve standardní programové nabídce na Linuxových systémech. Výsledky měření jsou zaneseny do grafu 3.6. Při zkoumání jednotlivých záznamů lze vypozorovat, že implementace FFmpeg používá průměrně o 10x více paměti, než u zbylých dvou implementací. V tabulce 3.4 je uvedeno paměťového

Obrázek 3.4: Testovací obrázek, který byl vybrán pro svoji různorodost sousedních pixelů.



Obrázek 3.5: Srovnání výkonu enkódování na obrázkem 3.4 v různých rozlišeních. V ose x je rozlišení obrazu, v ose y je čas na 1 pixel

srovnání velikostí výstupního formátu. V nejvyším měřeném rozlišení je patrné, že výstupní formát má čtyřnásobnou velikost výstupního souboru než implementace OpenJPEG. V porovnání s implementací FFmpeg je výsledná velikost souboru srovnatelná.

3.6 Shrnutí

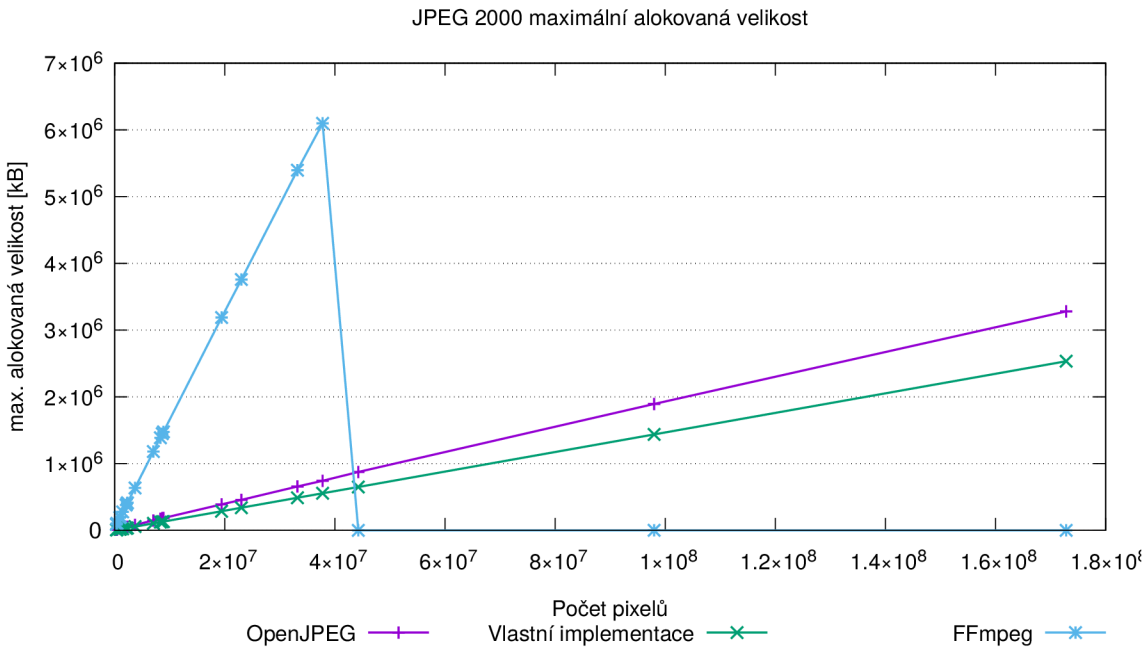
V rámci měření bylo zjištěno, že navzdory použité optimalizované knihovně pro diskrétní vlnkovou transformaci, není enkodér srovnatelelně výkoný jako ukázková implementace OpenJPEG nebo implementace formátu v FFmpeg. Může to být způsobeno neoptimálním převáděním reprezentace *float* čísla na datový typ *int32* pro potřeby aritmetického kodéru.

Slovní reprezentace	Šírka	Výška
20k	20240	8538
19k	19240	8116
17k	17240	7272
15k	15240	6429
14k	14240	6007
12k	12240	5163
10k	10240	4320
8k	8192	4608
8k UHD	7680	4320
IMAX	5616	4096
6k	6144	3160
4k DCI	4096	2160
DCI 3k	3996	2160
DCI 4k	4096	1714
QHD	2560	1440
4k UHD	3840	2160
WUXGA	1920	1200
DCI 2k	2048	1080
FHD	1920	1080
HDP	1600	900
HD	1360	768
HDTV	1280	720
DVPAL	720	576
DVNTSC	720	480
VGA	640	480

Tabulka 3.3: Tabulka testovaných rozlišení obrazu

Jediným kladem může být menší paměťová stopa při procesu enkódování. Navíc u řešení FFmpeg nebylo na testovacím stroji možné enkódovat velká obrazová data z formátu BMP.

Co se týče výstupního formátu obrazu, byl využit co nejjednodušší způsob, jak komprimovaná data zapsat do souboru. Při dekompresi jsou dostupné všechny informace, pro zařazení datového segmentu v obraze, ať na úrovni komponenty obrazu, rozlišení, podpásma či bloku dat.



Obrázek 3.6: Srovnání pro maximální alokovaný paměťový blok při enkódování

Rozlišení	OpenJPEG	Vlastní řešení	FFmpeg
20 240 x 8 538	58,0	244	-
15 240 x 6 429	46,1	146,3	-
10 240 x 4 320	31,5	73,4	73,3
8 192 x 4 608	25,5	61.9	58,5
7 679 x 4 320	23,1	54,8	52,3
5 616 x 4 096	16,4	38,3	36,8
6 144 x 3 160	14,8	33,0	33,1
4 096 x 2 160	8,9	16,6	18
3 996 x 2 160	8,9	16,3	17,7
4 096 x 1 714	8,3	15,5	16,9
3 839 x 2 160	6,3	12,6	13,5
2 560 x 1 440	4,3	7,4	7,8
1 920 x 1 200	3,0	4,8	5,0
2 048 x 1 080	2,6	4,5	4,5
1 920 x 1 080	2,6	4,3	4,4
1 600 x 900	1,9	3,0	3,0
1 360 x 768	1,3	2,1	2,0
1 280 x 720	1,3	2,0	2,0
720 x 576	0,58	0,91	0,86
720 x 480	0,49	0,76	0,71
640 x 480	0,43	0,67	0,63

Tabulka 3.4: Tabulka velikosti výstupního formátu dat v MB.

Kapitola 4

Závěr

V rámci bakalářské práce jsem se seznámil s možnostmi kódování kompresního formátu JPEG 2000. V rámci tohoto textu jsou rozebrány možnosti kódování z první a zároveň základní části standardu. Dále jsem prozkoumal již implementované řešení FFmpeg a OpenJPEG. V návaznosti na to byl vytvořen prototyp uvnitř implementace OpenJPEG, který obměnil způsob předzpracování, a bylo obměněno DWT voláním externí sdílené knihovny. Implementace byla prováděna iterativně, kdy se sledovala shodnost výstupního obrazu před úpravou a po úpravě. V další iteraci byla převzata implementace předzpracování z prototypu a vytvořen samostatný program pro kódování obrazu do vlastního formátu podobného JPEG 2000. Nakonec byl enkodér testován na výkon v podobě spotřebovaného času na jeden pixel a bylo zároveň naměřeno paměťové vytížení programu. Výsledky měření byly srovnány s implementacemi FFmpeg a OpenJPEG. Výsledkem práce je implementace, která je schopná zpracovávat vyšší rozlišení oproti FFmpeg a v nejvyšší kvalitě srovnatelná velikostí výstupního souboru s FFmpeg.

Jako příležitost pro další vývoj lze shledávat následující kroky. V prvé řadě by bylo možné implementovat vlastní aritmetický kodér, který by akceptoval na vstupu koeficienty v datovém typu float. Nebylo by poté potřeba transformovat každý blok na požadovanou strukturu. Tím by se mohl opravit neduh aktuální implementace v horším výkonu než ostatní testované implementace. U DWT by bylo možné implementovat zpracování pomocí reverzivní transformace využívají vlnku 5/3. Enkodér by tak mohl podporovat bezeztrátovou variantu JPEG 2000. Další inspirace pro možné vylepšení jsou v teoretické části tohoto textu.

Seznam použitých zkratek

JPEG - Joint Photographic Experts Group:
skupina stojící za stejnojmenným formátem

ISO - International Organization for Standardization:
Mezinárodní organizace pro normalizaci

EIC - International Electrotechnical Commission:
Mezinárodní elektrotechnická komise

DWT - Discrete Wavelet Transform:
Diskrétní vlnková transformace

ROI - Region of interests:
oblast zájmu

VLSI - Very-large-scale integration:
výroba polovodičových integrovaných obvodů ve velkém měřítku

CT - Computed Tomography:
vyšetřovací metoda využívající rentgenového záření k sledování vnitřností těl

RCT - Reversible color transform:
reverzní transformace barevného modelu

ICT - Ireversible color transform:
ireverzibilní transformace barevného modelu

CDF - Cohen-Daubechies-Feauveau:
trojice autorů stojící za analytickým vlnkovým filtrem 9/7

EBCOT - Embedded Block Coding:
kódovací proces v JPEG 2000

RLC - Run-Length coding:
komprese posloupnosti stejných dat do kompaktní reprezentace

SPP - Significance Propagation Pass:
propagace znaménka do další úrovně zpracování

MRP - Magnitude Refinement Pass:
propagace magnitud do další úrovně zpracování

CUP - Cleanup pass:
fáze úklidu kodéru

LPS - Least Probable Symbol:
nejméně pravděpodobný symbol

MPS - Most Probable Symbol:
nejvíce pravděpodobný symbol

YUV:
tříprvkový barevný model, jasová a dvě barevnonosné složky

Literatura

[1] Acharya, T.; Tsai, P.-S.: *JPEG2000 standard for image compression*. Hoboken: John Wiley, 2005, ISBN 0471484229.

[2] Bařina, D.; Zemčík, P.: Multimédia. 2013, (studijní opora, FIT VUT v Brně).

[3] Bařina, D.; Zemčík, P.: Lifting scheme cores for wavelet transform. 2015, (technická zpráva).

[4] Group, I.-T. S.: ISO/IEC 10918-1 Information Technology- Digital compression and coding of continuous-tone still images: Requirements and guidelines. 1994, [standard].

[5] Group, I.-T. S.: ISO/IEC 15444-1 Information Technology JPEG 2000 Image Coding System: Core Coding System. 2002, [standard].

[6] JPEG: Overview of JPEG 2000. [Online; navštíveno 5. 5. 2017].
URL https://jpeg.org/jpeg2000/

[7] Murray, J. D.; VanRyper, W.: *Encyclopedia of graphics file formats*. Sebastopol: O'Reilly & Associates, 1994, ISBN 1565920589.

[8] Taubman, D.: High performance scalable image compression with EBCOT. *IEEE Transactions on Image Processing*, ročník 9, č. 7, 2000: s. 1158–1170, ISSN 10577149, doi:10.1109/83.847830.

[9] Taubman, D.: Embedded block coding in JPEG 2000. *Signal processing. Image communication*, ročník 17, č. 1, 2002: s. 49–72, ISSN 09235965, doi:10.1016/S0923-5965(01)00028-5.

[10] Taubman, D. S.; Marcellin, M. W.: *JPEG2000*. New York: Springer, první vydání, 2002, ISBN 079237519X.

[11] Yoon, K.; Bae, S.; Choi, S.; aj.: The lifting-based DWT filter hardware design for JPEG2000. In *SoC Design Conference, 2008. ISOCC '08. International*, ročník 01, IEEE, 2008, ISBN 9781424425983, s. I–160–I–163, doi:10.1109/SOCDC.2008.4815597.