Dokumentace k projektu do předmětu MUL

Enkodér JPEG

Ondřej Pavela

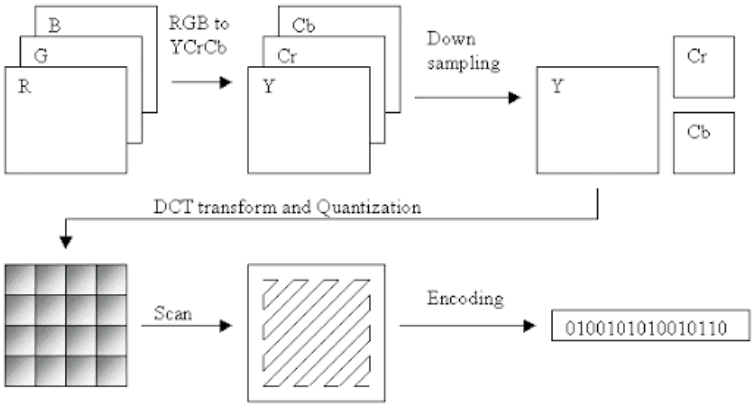
24. března 2021

1 Úvod

Cílem projektu bylo nastudovat kompresní formát JPEG a s tím související souborový formát JFIF sloužící pro uložení takto komprimovaných obrázků. Druhým cílem projektu poté byla samotná implementace JPEG enkodéru na základě poznatků z první části. Pro jednoduchost a rozšířenost byl zvolen *baseline* kompresní postup a z *extended* procesu byla implementována pouze podpora pro více Huffmanových tabulek.

2 Stručný popis *baseline* kompresního postupu

Komprese u *baseline* postupu probíhá v několika krocích, které jsou znázorněny na obrázku 1. Jednotlivé kroky budou nyní stručně popsány.



Obrázek 1: Kroky *baseline* procesu

2.1 Separace složek a konverze barevného modelu

Při JPEG kompresi se pracuje s jednotlivými obrazovými složkami odděleně a v prvním kroku je tedy potřeba tyto složky separovat do tří bufferů. To znamená průchod přes všechny pixely vstupního obrazu, zahození případného alfa kanálu (JPEG nepodporuje alfa kanál) a postupné ukládání 8 bitových R, G a B složek do připravených bufferů.

Kromě toho je při separaci potřeba také převést RGB barevný model na YC_bC_r, se kterým se při kompresi pracuje. Tato konverze se provádí pomocí následujících vzorců:

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= -0.1687R - 0.3313G + 0.5B + 128 \\ C_r &= 0.5R - 0.4187G - 0.0813B + 128 \end{aligned} \tag{1}$$

Prakticky ovšem následná *Diskrétní Kosinova Transformace* (DCT) při použití v JPEG kompresi vyžaduje, aby měl rozsah vstupních hodnot střed v nule [5]. Při převodu na YC_bC_r model se tedy ještě odečte konstantní hodnota 128 od každé složky, čímž zaručíme, že všechny hodnoty budou v rozsahu $[-128, 127]$. Tento posun o konstantu má ovšem při DCT vliv pouze na tzv. stejnosměrnou složku DC a ekvivalentní úprava by tedy zahrnovala odečtení hodnoty 1024 od výsledného DC koeficientu s indexy $[0, 0]$ po provedení transformace.

2.2 Podvzorkování chromatických složek a rozdělení na 8x8 bloky

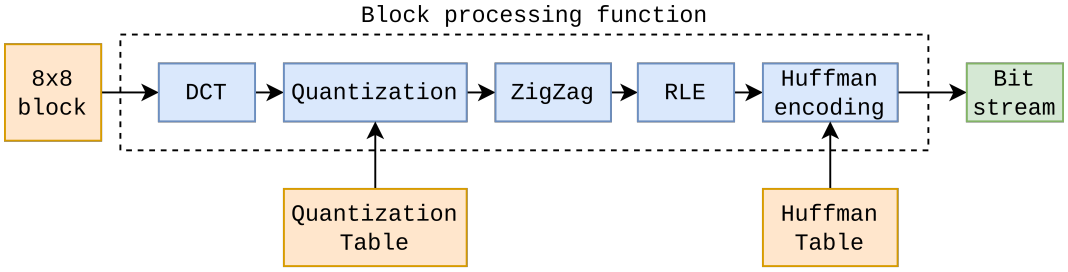
Druhým nepovinným krokem je podvzorkování chromatických složek C_b a C_r. Lidské oko je méně citlivé na změny barev než na změny jasu a proto se podvzorkovávají pouze chromatické složky, jejichž degradace v kvalitě není tak viditelná ve finálním obrazu. Podporovány jsou dva režimy podvzorkování: 4:2:0 (v horizontálním i vertikálním směru na polovinu) a 4:2:2 (v horizontálním směru na polovinu).

Po případném podvzorkování následuje rozdělení jednotlivých složek na tzv. MCU (Minimum Coded Unit) [3] bloky o velikosti 8x8 pixelů. Tyto bloky jsou poté dále zpracovávány individuálně. V případě, že původní obraz neměl rozměry v násobcích 8, pak musí být patřičně rozšíření, například duplikací posledního sloupce, či řádku v jednotlivých složkách. Pokud byly navíc podvzorkovány chromatické složky, pak musí být jasová složka Y rozšířena v jednom nebo obou směrech na násobek 16.

Důvod je ten, že při samotném kódování je nejprve zakódován 1, 2 nebo 4 bloky jasové složky (podle použitého režimu podvzorkování) a poté vždy následuje zakódovaný jeden blok složek C_b a C_r. Aby toto mohlo být dodrženo, musí být počet bloků jasové složky v jednom nebo obou směrech roven násobku 2, tzn. násobku 16 v rozměrech původního obrázku.

2.3 Zpracování jednoho bloku

Po separaci složek a rozdělení na bloky následuje zpracování samotných bloků. Vždy 1 až 4 bloky jasové složky Y a poté jeden blok složky C_b a jeden blok složky C_r. Proces zpracování jednoho bloku zobrazený na obrázku 2 se pro jednotlivé složky neliší. Jediný rozdíl je, že pro jasovou složku je



Obrázek 2: Proces zpracování jednoho bloku

použita jiná kvantizační a Huffmanova tabulka (vstupy funkce), než pro barevné složky.

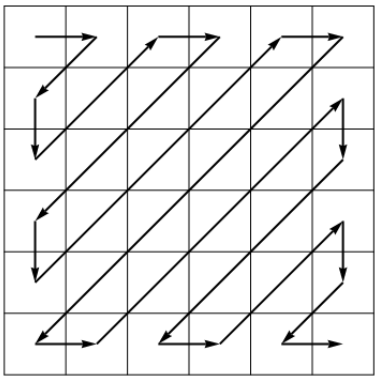
Prvním krokem je provedení `DCT` transformace nad vstupním blokem o velikost 8x8. Výsledkem je blok koeficientů o stejné velikosti. Následuje kvantizace tohoto bloku, což je hlavním zdrojem ztráty kvality (DCT samo o sobě nezpůsobuje ztrátu kvality). Ke kvantizaci se použije vstupní kvantizační tabulka o velikosti 8x8, která obsahuje čísla, kterými podělíme odpovídající koeficienty. Po tomto kroku máme blok 8x8 celých čísel v rozmezí $[-2047, 2047]$. Obrázek 3 obsahuje standardní

Quantization Table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Obrázek 3: Standardní kvantizační tabulka pro jasovou složku Y (50% kvalita)

tabulku pro jasovou složku Y [4]. Následuje linearizace tzv. *ZigZag* způsobem, který je znázorněn na obrázku 4. Linearizací získáváme z 2D bloku dat 1D sekvenci čísel. Prakticky se kvantizační a linearizační krok provádí zároveň.



Obrázek 4: Schéma ZigZag linearizace

Nultý DC koeficient se kóduje rozdílově oproti DC koeficientu předchozího bloku stejné složky, tzn. $d = DC^Y - DC^Y_{previous}$. Samotný rozdíl d se zakóduje pomocí čísla kategorie a indexu v rámci dané kategorie. Kategorií je celkem 11 a jejich rozsahy jsou navzájem disjunktní, pro příklad 1. kategorie má rozsah $\langle -1, 1\rangle$, druhá poté $\langle -3, -2\rangle \cup \langle 2, 3\rangle$ a tak dále. Číslo kategorie je kódováno pomocí Huffmanova kódu počtu bitů, které jsou potřeba pro reprezentaci indexu, tzn. pomocí pseudokódu to lze vyjádřit následovně: `HuffmanTable[bitCount(index)]`. Převod rozdílu d na index je v podstatě proces vyjádření znaménkového čísla d pomocí kódu s posunutou nulou, kde nulu v tomto případě tvoří poslední záporné číslo z dané kategorie. Například indexy v 2. kategorii jsou kódovány následovně: $-3 = 00_b$, $-2 = 01_b$, $2 = 10_b$, $3 = 11_b$, kde -2 odpovídá posunuté nule. Jak je vidět,

pro kladné hodnoty d je index roven d. Nulový rozdíl $d = 0$ se kóduje pouze pomocí Huffmanova kódu nuly.

Zbývajících 63 AC koeficientů se kóduje podobným způsobem, akorát se zde používá pouze 10 kategorií (celkový rozsah možných hodnot je menší, $\langle -1023, 1023 \rangle$) a nekóduje se rozdíl, ale přímo hodnota koeficientu. Kromě toho se do horních 4 bitů klíče do Huffmanovy tabulky vkládá číslo udávající počet nul, které předcházely koeficientu v linearizované sekvenci, aby se ušetřilo místo, tzn. `HuffmanTable[(zeros << 4) + bitCount(index)]`. V podstatě se jedná o upravené *Run-Length Encoding* (RLE) kódování. Pokud se v sekvenci nachází 16 nul po sobě, pak se tento fakt kóduje pomocí Huffmanova kódu speciální značky `0xF0`. Pokud je poslední nenulový koeficient před koncem sekvence, pak se kódování bloku ukončí pomocí Huffmanova kódu speciální značky EOB (End of Block) s hodnotou `0x00`. Podrobnější informace o kódování lze nalézt ve studijní opoře do předmětu Multimédia [1].

3 Implementace

Celý enkodér byl implementován v jazyce C++17. Pro načtení obrazových dat v 24 bitovém RGB formátu byla použita knihovna `STB` (https://github.com/nothings/stb), která podporuje rozumnou škálu formátů.

Po načtení dat jsou nejprve separovány barevné složky do samostatných bufferů a zároveň je provedena transformace do YC_bC_r barevného modelu pomocí vzorců uvedených v předchozí sekci (od hodnot je ještě navíc odečtena hodnota 128, tzv. *level shift*). Po separaci jsou rozměry jednotlivých složek rozšířeny na odpovídající násobky (8 nebo 16) podle zvoleného režimu podvzorkování. Rozšíření je provedeno kopírováním hodnot posledního sloupce či řádku. Po rozšíření následuje samotné podvzorkování (pokud je zapnuto). Konkrétně je podvzorkování implementováno výpočtem průměrné hodnoty ze dvou nebo čtyř sousedních hodnot barevných složek.

Poté již následuje rozdělení složek na bloky o velikosti 8x8. Prakticky se pouze jedná o reorganizaci dat v pomocných bufferech stejné velikosti. Rozdělení je realizováno postupným průchodem přes všechny řádky jedné složky, přičemž každý řádek je rozštěpen na 8 bytové sekvence, které jsou uloženy na odpovídající místa do pomocného bufferu.

Uvedené kroky by bylo možné provádět do určité míry současně, nicméně z důvodu lepší čitelnosti kódu jsou prováděny odděleně i za cenu ztráty určité efektivity, jelikož je potřeba několik průchodů daty místo jednoho.

3.1 Hlavní smyčka zpracování

Po předzpracování vstupních dat následuje samotné zpracování jednotlivých bloků. K dispozici jsou dva módy: sekvenční a paralelní vícevláknový. Sekvenční pracuje způsobem, který byl popsán v předchozí sekci, tedy vždy jsou zpracovány 1 až 4 bloky (podle zvoleného režimu podvzorkování) světlonosné složky Y a poté 1 blok od každé barevné složky. Tento proces se opakuje, dokud nejsou zakódovány všechny bloky. Zjednodušenou verzi kódu, který řídí zpracování bloků je možné vidět na následujícím úryvku 1.

Funkce pro zpracování bloků poté využívají různé Huffmanovy a kvantizační tabulky podle toho, o jakou složku se jedná. Výsledkem zpracování je buffer s ukazateli na struktury `ElementCode`, které udržují informaci o hodnotě a počtu bitů, které jsou potřeba pro uložení dané hodnoty. Tyto struktury jsou fyzicky uloženy ve statických Huffmanových tabulkách a tabulce s předpřipravenými

```
for (uint y = 0; y < BlocksY; y += verticalStep) {
    for (uint x = 0; x < BlocksX; x += horizontalStep) {

        /* Process 1, 2 or 4 luminance blocks */
        for (uint i = 0; i < verticalStep; i++) {
            for (uint j = 0; j < horizontalStep; j++) {
                ProcessBlock(yBlocks[y+i][x+j]);
            }
        }

        if (rgb) {
            /* Process chrominance blocks */
            ProcessBlock(CbBlocks[y+i][x+j]);
            ProcessBlock(CrBlocks[y+i][x+j]);
        }
    }
}
```

Listing 1: Zjednodušený kód hlavní smyčky zpracování

kódy pro všechny možné hodnoty AC koeficientů a rozdílů DC koeficientů. Tato tabulka má velikost 4096 položek, jelikož rozsah možných hodnot je $[-2047, 2047]$.

Po zakódování všech bloků je proveden jeden průchod přes tento buffer, hodnoty ukazatelů jsou dereferencovány a všechny `ElementCode` struktury jsou vloženy do bitového bufferu `BitBuffer`, který je zodpovědný za vytvoření finálního kompaktního bitového toku, který může být uložen do souboru.

Důvod, proč nejsou hodnoty ukládány přímo do bitového bufferu, ale nejprve do pomocného bufferu s ukazateli je jednoduchý. Finální program podporuje i vytvoření optimálních Huffmanových tabulek na základě vstupních dat. Abychom však mohli vytvořit optimální tabulky, je nejprve potřeba vytvořit histogramy četností jednotlivých kódů, tedy sesbírat statistiky. Z toho důvodu jsou technicky potřeba dva průchody. Jeden pro zjištění četností a druhý pro samotné zakódování. Pro urychlení tohoto procesu jsou hodnoty svým způsobem zakódovány již v prvním průchodu, kdy si ukládáme ukazatele na hodnoty, které však budou k dispozici až po vytvoření optimálních tabulek. Cenou za tento přístup je jistá neefektivita v případě, kdy uživateli postačují výchozí empiricky určené neoptimální tabulky, které jsou k dispozici již při prvním průchodu. Tyto tabulky jsou spolu s výchozími kvantizačními tabulkami uvedeny v JPEG standardu [4].

Paralelní vícevláknové zpracování probíhá obdobným způsobem s tím rozdílem, že složky jsou rozděleny na horizontální pásy podle počtu vláken a každé vlákno poté zpracovává pouze odpovídající bloky všech složek. Druhý drobný rozdíl spočívá v tom, že následující vlákna nejsou schopna dopočítat rozdíly DC koeficientů pro své první bloky, protože DC koeficienty posledních bloků předchozího vlákna ještě nejsou známy. Z toho důvodu jsou ukládány DC koeficienty prvních a posledních bloků do pomocných bufferů, spolu s indexy, kam mají být po dopočítání zakódované hodnoty vloženy. Jakmile všechna vlákna dokončí zpracování svých bloků, jsou jejich buffery při sekvenčním průchodu sloučeny do jednoho bitového toku a na odpovídající místa jsou vloženy dopočítané rozdíly DC koeficientů.

3.2 Zpracování jednoho bloku

Funkce pro zpracování jednoho bloku v podstatě odpovídá diagramu na obrázku 2 s tím rozdílem, že neprodukuje přímo bitový tok, ale pouze seznam ukazatelů. Po provedení `DCT` transformace je provedena *ZigZag* linearizace, při které jsou hodnoty koeficientů zároveň kvantizovány, tedy oba kroky jsou prováděny zároveň. Při linearizaci je také uložen index poslední nenulové hodnoty, abychom věděli, kdy můžeme kódování bloku ukončit a vložit *EOB* značku. Pro jednoduchost a rychlost jsou linearizační indexy uloženy v pomocném poli a není tak nutné realizovat *ZigZag* průchod složitým cyklem s podmínkami. Samotné kódování nenulových koeficientů je prováděno způsobem, který byl popsán v sekci 2.3. Navíc jsou ještě v průběhu inkrementovány hodnoty v histogramech, které jsou poté použity při výpočtu optimálních Huffmanových tabulek.

Pro samotnou `DCT` transformaci byla použita knihovna *FFTW* (Fastest Fourier Transform in the West). Tato knihovna ovšem počítá 1D `DCT-II` transformaci pomocí následujícího vzorce:

$$Y_k = 2\sum_{n=0}^{N-1} X_n \cos\left[\frac{k\pi}{N}\left(n+\frac{1}{2}\right)\right],$$

který je ovšem lehce odlišný od vzorce, který se používá při JPEG kompresi:

$$Y_k = \sum_{n=0}^{N-1} X_n \lambda_k \sqrt{\frac{2}{N}} \cos\left[\frac{k\pi}{N}\left(n+\frac{1}{2}\right)\right],$$

kde $\lambda_k = \frac{1}{\sqrt{2}}$ pro $k = 0$, jinak $\lambda_k = 1$. Z toho důvodu byly všechny potřebné hodnoty $\lambda_k\sqrt{\frac{2}{N}}$ (rozšířené pro 2D transformaci) předpočítány a po provedení FFTW `DCT` transformace jsou výsledné koeficienty těmito hodnotami násobeny.

Kromě toho byla implementována i vlastní verze separabilní 2D `DCT` transformace s využitím předpočítaných koeficientů a SIMD akcelerace. Tato verze byla při vypnutých optimalizacích zhruba dvakrát pomalejší, než verze z FFTW knihovny.

3.3 Generování optimálních Huffmanových tabulek

Základní *greedy* algoritmus pro generování optimálních Huffmanových tabulek je vcelku jednoduchý a je popsán například ve studijní opoře do předmětu Multimédia [1]. Co už se ovšem většina autorů všemožných článků o JPEG kompresi neobtěžuje adresovat je fakt, že tento algoritmus generuje optimální kódy neomezené bitové délky, což je však naprosto nepoužitelné pro JPEG kompresi, kdy všechny kódy musí být maximálně 16 bitů dlouhé.

Upravený algoritmus založený na heuristice, který generuje kódy omezené délky je uveden v JPEG standardu [4] a je zde popsán pomocí několika vývojových diagramů. Tento algoritmus bohužel není optimální a může generovat i tabulky, které nevedou na maximální možnou kompresi. V drtivé většině případů je ovšem rozdíl ve velikosti naprosto minimální a v praxi se proto tento algoritmus používá, jelikož je dostatečně implementačně jednoduchý a relativně výpočetně nenáročný na rozdíl od optimálního algoritmu.

Ve finálním programu byla proto implementována tato sub-optimální verze a při implementaci byla jako reference použita také knihovna *libjpeg-turbo* (`https://libjpeg-turbo.org`), kde je tento algoritmus implementován. Jediný rozdíl ve finální implementaci je způsob, jakým jsou postupně vyhledávány prvky s nejnižšími četnostmi. Pro tyto účely byla použita datová struktura *min-heap*, pro urychlení vyhledávání.

3.4 Vytvoření `JFIF` souborů

Posledním krokem je obalení komprimovaného datového toku hlavičkou a patičkou. Patička obsahuje pouze Huffmanův kód speciální značky *EOI* (End of Image) s hodnotou `0xD9`, přičemž poslední byte obrazových dat je doplněn jedničkovými bity. Hlavička JFIF souboru je již mnohem komplikovanější a pokud se chce člověk vyhnout podrobnému čtení standardu, pak je dohledání úplných a korektní informací na internetu poměrně náročné. Poměrně dobrým referenčním zdrojem s informacemi ve stravitelné podobě byla následující stránka [2].

Úplně nejlepším zdrojem ovšem byly zdrojové kódy jednoduchého `TooJpeg` JPEG enkodéru od Stephana Brummeho dostupné na následující adrese https://github.com/stbrumme/toojpeg. Na základě tohoto kódu jsem zjistil, v jakém pořadí, v jakém formátu a s jakými speciálními značkami je potřeba zapsat do hlavičky jednotlivé tabulky a různá metadata.

4 Kompilace a spuštění

Pro sestavení projektu je potřeba `CMake`, `C++` překladač s podporou `C++17` a knihovna `FFTW`, která musí být v nainstalována v systémových cestách tak, aby ji `CMake` nalezl. Dále je potřeba knihovna `stb_image`, která je ovšem přiložena k zdrojovým kódům projektu. Pak by mělo být možné projekt zkompilovat standardním způsobem jako jakýkoliv jiný projekt nad `CMake`.

Nápovědu ke spuštění programu poté poskytuje následující úryvek:

```
[Usage] JPEG_encoder filepath [-rgb] [-optimize] [-subsample VAL] [-q VAL] [-o PATH]
Options:
    filepath           Path to the input image.
    -rgb               Output RGB image.
    -optimize          Calculate optimized Huffman tables.
    -mt                Multi-threaded compression.
    -subsample VALUE   Subsample chrominance component.
                       Possible values are '4:2:0' (2x horizontal + vertical)
                       and '4:2:2' (2x horizontal only).
    -q VALUE           JPEG Quality level, possible values in range <1, 100>.
    -o FILEPATH        Specify output path for JPEG image.
    -h | --help        Show usage.
```

Prvním parametrem musí být cesta k libovolnému obrázku ve formátu, který podporuje knihovna `stb_image` (8 bitů na složku). Dále je možné zvolit, zda si uživatel přeje barevný RGB nebo *grayscale* výstup. Pokud je vstupní obrázek v *grayscale*, pak je přepínač pro RGB výstup ignorován. Dále je možné pomocí přepínače `-optimize` zapnout optimalizaci Huffmanových tabulek pro vyšší míru komprese. Přepínač `-mt` slouží k zapnutí paralelního vícevláknového režimu. K zapnutí podvzorkování a výběru konkrétního režimu pak slouží přepínač `-subsample`. Výběr kvality výstupního obrazu (úprava výchozí kvantizační tabulky) je řízena přepínačem `-q`, který má povolené hodnoty v rozsahu $[1, 100]$. Pokud si uživatel přeje uložit výstupní obrázek na konkrétní místo v souborovém systému, pak lze využít přepínače `-o`.

5 Výsledky

Výsledný program byl otestován na několika obrázcích a byla porovnána výkonnost s dříve zmíněnou knihovnou `TooJpeg`. Konkrétně byl výkon testován na obřím obrázku BlueMarble od NASA. Tento obrázek má rozměry 21600x10800 pixelů a v PNG formátu zabírá přibližně 197 MB.

Knihovna `TooJpeg` stihla zpracovat tento obrázek za cca 2745 milisekund a má paralelní implementace za zhruba 5030 milisekund. Výsledný program je tedy zhruba dvakrát pomalejší oproti knihovně `TooJpeg`. Hlavní důvody jsou dva. Za prvé zmíněná knihovna využívá specializovaný algoritmus pro výpočet `DCT` transformace, který je optimalizován přímo pro obrazová data: A Fast DCT-SQ Scheme for Images a za druhé je má implementace separace, podvzorkování a blokování vstupních dat značně neefektivní a způsobuje poměrně vysokou spotřebu paměti RAM. Samotná výpočetní fáze bez tohoto předzpracování pak trvala zhruba pouhých 2100 milisekund.

Na závěr byl otestován vliv optimalizace Huffmanových tabulek na celkovou velikost výsledného komprimovaného obrázku. Výsledky je možné vidět v tabulce 1. Byly otestovány různé kombinace podvzorkování s 50% a 100% kvalitami na obrázku o velikosti 3840x2160 pixelů. Jak je vidět,

Obrázek 3840x2160p			
Režim	Bez optimalizace (B)	S optimalizací (B)	Úspora %
Q100 + 4:2:0	2 960 960	2 453 315	17.14
Q50 + 4:2:0	607 933	584 891	3.79
Q100 + 4:2:2	3 512 908	2 990 034	14.88
Q50 + 4:2:2	662 422	630 190	4.87
Q100 + 4:4:4	4 687 714	4 073 871	13.09
Q50 + 4:4:4	755 362	694 788	8.02

Tabulka 1: Vliv optimalizace Huffmanových tabulek na výslednou velikost

největší vliv má optimalizace tabulek při vyšších kvalitách, kdy při kvantizaci nedochází k vynulování tak velkého počtu koeficientů. Úspora nad 10% je pak poměrně znatelná a má smysl zejména u velkých obrázků.

Literatura

[1] Bařina, D.; Zemčík, P. : Multimédia - Studijní opora. online, Březen 2021. Dostupné z: <https://www.fit.vutbr.cz/~ibarina/pub/mul-opora-2021-03-22.pdf>

[2] DCube-Software-Technologies : JPEG File Layout and Format. online, Červenec 2002. Dostupné z: <http://vip.sugovica.hu/Sardi/kepnezo/JPEG%20File%20Layout%20and%20Format.htm>

[3] Hass, C. : JPEG Minimum Coded Unit (MCU) and Partial MCU. online, 2018. Dostupné z: <https://www.impulseadventure.com/photo/jpeg-minimum-coded-unit.html>

[4] ITU : ISO/IEC 10918-1 : 1993(E) CCIT Recommendation T.81. 1993. Dostupné z: <http://www.w3.org/Graphics/JPEG/itu-t81.pdf>

[5] Wikipedia : JPEG. online, Březen 2021. Dostupné z: <https://en.wikipedia.org/wiki/JPEG>