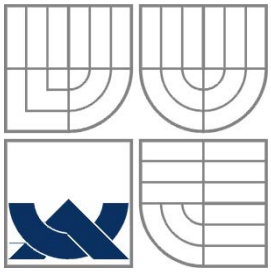
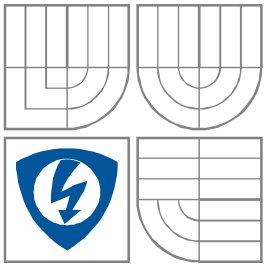


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV BIOMEDICÍNSKÉHO INŽENÝRSTVÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF BIOMEDICAL ENGINEERING

FORMÁTY OBRAZOVÝCH DAT
IMAGE FORMATS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE David Wolanský
AUTHOR

VEDOUCÍ PRÁCE Ing. Radovan Jiřík, Ph.D.
SUPERVISOR

BRNO, 2008

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami:

1. Pan/~~paní~~

Jméno a příjmení: David Wolanský
Bytem: 8. května 48, Šumperk, 787 01
Narozen/a (datum a místo): 11. října 1982 v Šumperku

(dále jen „autor“)

a

2. Vysoké učení technické v Brně

Fakulta elektrotechniky a komunikačních technologií
se sídlem Údolní 53, Brno, 602 00
jejímž jménem jedná na základě písemného pověření děkanem fakulty:
prof. Dr. Ing. Zbyněk Raida, předseda rady oboru Elektronika a sdělovací technika
(dále jen „nabyvatel“)

Čl. 1

Specifikace školního díla

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

   ☐ disertační práce
   ☐ diplomová práce
   ☒ bakalářská práce
   ☐ jiná práce, jejíž druh je specifikován jako .....
   (dále jen VŠKP nebo dílo)

   Název VŠKP: Formáty obrazových dat
   Vedoucí/ školitel VŠKP: Ing. Radovan Jiřík, Ph.D.
   Ústav: Ústav biomedicínckého inženýrství
   Datum obhajoby VŠKP: _____

   VŠKP odevzdal autor nabyvateli*:

   ☒ v tištěné formě – počet exemplářů: 2
   ☒ v elektronické formě – počet exemplářů: 2

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.

3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.

4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

* hodící se zaškrtněte

Článek 2

Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizovaní výpisů, opisů a rozmnoženin.

2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.

3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti

   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)

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

Článek 3

Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.

2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.

3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.

4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: 6. června 2008

……..	……
Nabyvatel	Autor

Abstrakt

Tato práce se zabývá podrobnou analýzou grafického formátu GIF. Dělí se na dvě základní části, teoretickou a praktickou. Teoretická část zahrnuje důkladný rozbor interní strukturu souboru, tj. popis jednotlivých typů bloků, ze kterých se soubor skládá. Charakteristika každého typu bloku představuje popis vlastností a detailní popis jeho jednotlivých položek. Součástí je i stručná charakteristika nejpoužívanějších kompresních metod aplikovaných v oblasti komprese obrazových dat.

Praktická část zahrnuje návrh a realizaci výukové aplikace pro přehledné zobrazení interní struktury souboru vybraného obrázku typu GIF a jeho parametrů. Realizace aplikace spočívá v tvorbě algoritmů pro selekci jednotlivých typů bloků a v konstrukci grafického rozhraní aplikace.

Klíčová slova

Kompresní metody, GIF, bloky, barvová paleta, rozšíření, rámec, struktura souboru, animace, rozhraní

Abstract

The thesis deals with a detailed analysis of the GIF file format. It is divided into two basic parts, theoretical and practical. The theoretical part includes a thorough analysis of the internal file structure, i.e. the description of the individual block types, of which the file consists. Each block-type characteristic includes properties and a detailed description of its individual items. This part also includes a brief description of the most common compression methods applied in the field of image compression.

The practical part includes the design and implementation of an edutainment application for synoptic viewing of the internal structure for a chosen GIF file and its parameters. The implementation of the application covers algorithms for selection of individual block types and realization of a graphical user interface.

Keywords

Data compression methods, GIF, blocks, colour table, extension, frame, file structure, animation, interface

Bibliografická citace

WOLANSKÝ, D. *Formáty obrazových dat: bakalářská práce.* Brno: FEKT VUT v Brně, 2008. 46 s., 1 příl.

Prohlášení

Prohlašuji, že svou bakalářskou práci na téma Formáty obrazových dat jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této bakalářské práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne 6. června 2008 …….

podpis autora

Poděkování

Děkuji vedoucímu bakalářské práce Ing. Radovanu Jiříkovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce.

V Brně dne 6. června 2008 ……..

podpis autora

Obsah

Úvod

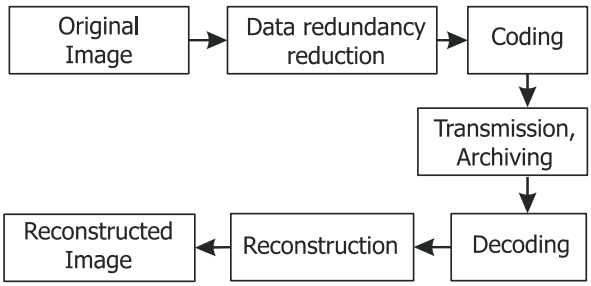
Hlavní náplní této práce je návrh a realizace výukové aplikace pro přehledné zobrazení interní struktury souboru typu GIF. Aplikace je zamýšlena jako podpora výuky v oblasti počítačové grafiky.

Nezbytným teoretickým předpokladem pro správný návrh celé aplikace je důkladná analýza grafického formátu GIF. Součástí analýzy je nejprve popis jeho základních vlastností jako bitová hloubka, paleta barev, prokládání, animace atd. Velkou pozornost je pak třeba věnovat interní struktuře souboru typu GIF. Jelikož jsou interní data souboru tvořena tzv. bloky (např. hlavička, globální paleta barev atd.), je nutné se zaměřit především na jejich funkci a detailní obsah jednotlivých položek Velmi důležitou roli hraje i znalost hierarchie jednotlivých typů bloků v souboru. Tato znalost, jak se ukáže, je velmi důležitá z hlediska porozumění funkčnosti obrázku jako celku. Jelikož se práce zabývá grafickým formátem, je namístě zmínit se i o některých kompresních metodách, které nachází uplatnění v oblasti komprese obrazových dat.

Důkladný rozbor interní struktury souboru je nezbytným teoretickým předpokladem pro úspěšnou realizaci navrhované výukové aplikace. Požadavky na funkce aplikace jsou takové, aby co nejpřehledněji zobrazovala interní data souboru a zároveň umožňovala uživateli základní operace s vybraným obrázkem resp. souborem. Mezi tyto operace patří obecně jakákoli změna dat vybraného souboru, mazání dat a export a import dat mezi soubory. V programu MATLAB GUI je tedy nutné navrhnout rozhraní s vhodnými komponentami, které budou plnit požadované funkce. K detailnímu zobrazení interních dat souboru resp. jednotlivých typů bloků a jejich položek do komponent je nezbytné navrhnout příslušné algoritmy pro jejich detekci a uložení do proměnných. Lze tedy říci, že spojení návrhu jednotlivých algoritmů a vhodného rozhraní může vést k funkční výukové aplikaci, pomoci které si uživatel prohloubí znalosti o interní struktuře souboru typu GIF.

1 Komprese obrazových dat

Komprese dat je v současné době velmi účinný nástroj, který umožňuje pracovat (tj. přenášet, ukládat, upravovat apod.) s velkými objemy digitalizovaných dat i při velmi omezených technických možnostech používané výpočetní techniky (záznamová kapacita paměťových médií, přenosová kapacita telekomunikačních kanálů apod.). Pod pojmem komprese se tedy rozumí proces, který se používá pro zredukování fyzické velikosti bloku digitalizovaných dat.



Obr. 1.1. Komprese obrazu a jeho rekonstrukce

Jak je naznačeno na obr. 1.1, vstupní data jsou pomocí komprimátoru zkomprimována vhodným kompresním algoritmem a poté jsou uložena na záznamové médium (paměťový obvod, harddisk počítače apod.) nebo přenesena po telekomunikačním kanále (družicový spoj, telefonní linka apod.). Pro další následné zpracování jsou pak obrazová data dekomprimována pomocí dekomprimátoru. Na tomto místě je nutné podotknout, že pro úspěšné obnovení komprimovaných dat je bezpodmínečně nutné mít přesnou znalost použitého kompresního algoritmu.

Při volbě kompresních algoritmů se sleduje celá řadu důležitých parametrů. Jedním z nejdůležitějších parametrů je kompresní poměr [1]:

$$\kappa = \frac{n_1}{n_2} = \frac{velikost\_nekomprimovaného\_souboru}{velikost\_komprimovaného\_souboru} \tag{1.1}$$

a s tím související relativní úspora paměti:

$$R = \left(1 - \frac{1}{\kappa}\right) \cdot 100 \quad [\%]. \tag{1.2}$$

Pro další dělení jednotlivých kompresních metod se jako klíčový parametr volí ztrátovost použité metody. Z tohoto hlediska se tedy rozlišují

- Bezeztrátové kompresní metody
- Ztrátové kompresní metody

1.1 Bezeztrátové kompresní metody

Bezeztrátové kompresní metody (redukce redundance) jsou metody, kde nedochází ke ztrátě informace, tzn. nadbytečnou složku lze v přijímači obnovit. Redundance je určená statickými charakteristikami relevantní složky obrazu, přičemž dominantní postavení mají korelační vlastnosti. Pro potlačení redundance je nutné zjistit množství informace, obsažené v nějakém obraze, který je možné formálně popsat pojmem entropie. Čím má obraz větší „neurčitost“, tj. entropii, tím více informací obsahuje a k jeho záznamu je potřebných více bitů [2]. Budeme-li mít abecedu ***s***, jejíž symboly tvoří uvedené hodnoty,

$$\mathbf{s} = \{s_1, s_2, .., s_N\}. \tag{1.3}$$

Jednotlivé symboly abecedy se vyskytují s pravděpodobnostmi,

$$\mathbf{p} = \{p_1, p_2, .., p_N\}, \sum_{i=1}^{N} p_i = 1. \tag{1.4}$$

Můžeme tedy pro entropii abecedy **s** psát,

$$H(s) = \sum_{i=1}^{N} p_i \log_2 p_i \quad \text{[bitů/symbol]}. \tag{1.5}$$

Rozdíl entropie obrazu a velikosti jeho skutečného záznamového rozsahu *b* udává redundanci,

$$r = H(s) - b. \tag{1.6}$$

Ve většině případů obsahuje obraz obrazové body, vyznačující se vzájemnými statistickými vazbami, tj. vzájemně korelované obrazové body. Vyloučením těchto vazeb, tj. dekorelací obrazu je možné dosáhnout výrazného snížení počtu bitů, potřebných k jeho záznamu bez toho, aby poklesl jeho informační obsah. Typickou ukázkou bezeztrátových kompresních metod jsou [1]:

- Huffmanovo kódování (též entropické)
- Proudové kódování (RLC - Run Length Coding)
- Lempel-Ziv-Welchova metoda (slovníkový algoritmus)
- Prediktivní metody

1.1.1 LZW (Lempel-Ziv-Welch)

Základy slovníkových kompresních metod položili v roce 1977 pánové Abraham Lempel a Jakob Ziv. Jejich algoritmus se ujal pod názvem LZ77 a tvoří základ dnes tolik oblíbených kompresních programů jako ZIP a ARJ. O sedm let později byla původní kompresní metoda zdokonalena Terry Welchem a výsledkem byl univerzální komprimační algoritmus, dnes všeobecně označovaný jak LZW.

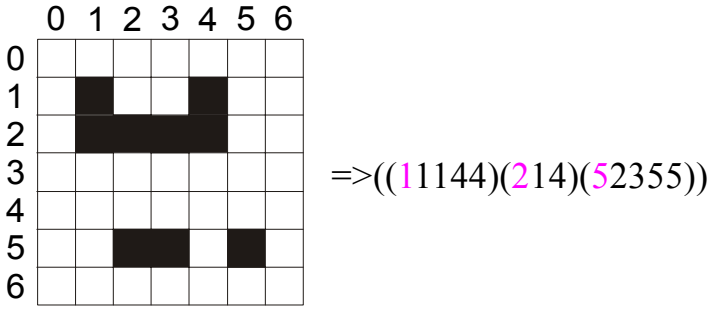
LZW je všeobecný kompresní algoritmus, který je schopen zpracovávat téměř jakýkoli typ dat. Nepracuje s pevně stanoveným datovým slovníkem, ale při každé kompresní úloze začíná tvořit nový datový slovník, specifický pro daný vstupní nekomprimovaný obraz.

Datové vzorky (podřetězce) jsou definovány jako toky dat a shodují se se vstupy do slovníku. Pokud se podřetězec ve slovníku nevyskytuje, je podle obsahu informace vytvořena kódová fráze a je uložena do slovníku. Fráze je poté zapsána do komprimovaného výstupního toku. Pokud se daný podřetězec v datech znovu objeví je fráze tohoto podřetězce už uloženého ve slovníku přímo zapsána na výstup. Protože hodnota fráze má fyzickou velikost, která je menší než původní podřetězec, je dosaženo datové komprese.

Lempel–Ziv–Welchův algoritmus je v současné době hojně používán v rozšířených grafických formátech TIFF a GIF. Také se vyskytují různé varianty LZW komprese. Populární je například varianta algoritmu, která sleduje kompresní proces tak, aby nedošlo k poklesu účinnosti komprese. Pokud k nějakému poklesu dojde, je nejméně používaná fráze ze slovníku odstraněna, čímž se vytvoří místo pro umístění nové, více se vyskytující fráze. Lze také zrušit a přebudovat kompletně celý slovník [1].

1.1.2 RLC (Run Length Coding)

Je to algoritmus založený na předpokladu, že obraz se skládá z oblastí s charakteristickou hodnotou jasu. Když při popisu obrazu následuje více shodných obrazových bodů za sebou mohou být efektivně zakódované posloupností dvojic vyjadřujících početnost obrazového bodu a jeho jas. Ve většině případů je toto kódování vykonáváno v rámci jednoho řádku. Jde tedy o kódování délek úseků řádků. Existuje několik modifikací RLC algoritmů, všeobecně však RLC algoritmus patří k nejjednodušším komprimačním metodám. Tento algoritmus dosahuje kompresního efektu rozšířením výstupní kódové abecedy. Pro obrazy, které jsou charakteristické velkým počtem vodorovných čar, je kódování tímto kódem velmi efektivní. V případě, že obraz obsahuje neopakující se obrazové prvky na sousedních pozicích, dochází k záporné kompresi, tj. k zvětšení výstupního souboru. Na obr. 1.2 je příklad RLC kódování. Výstupní posloupnost je charakterizována označením řádku (červené číslo) a hodnotou na určenou pozicí sloupce a řádku. RLC algoritmus bývá označovaný také jako proudové kódování [1].

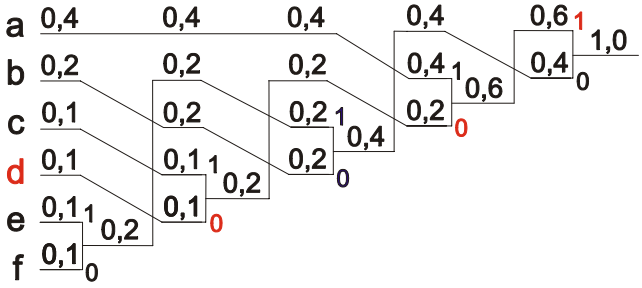


Obr. 1.2. Jednoduchý příklad RLC kódování

1.1.3 Huffmanovo kódování

Huffmanův kód se používá pro samostatné přímé bezeztrátové kódování, případně jako doplněk prediktivního kódování nebo jiných bezeztrátových metod. Metoda začíná vytvořením seznamu abecedních symbolů v sestupném pořadí jejich pravděpodobností. Pak se konstruuje strom, se symbolem v každém listu a to zdola nahoru. To se provádí v krocích, kdy se v každém kroku vyberou dva symboly s nejnižší pravděpodobností, sečtou se do vrcholu místního sbíhání větví, vyškrtnou se ze seznamu a nahradí se přidaným symbolem,

který je oba reprezentuje. Když se strom redukuje na jediný přidaný symbol (který reprezentuje celou abecedu), je strom dokončen. Strom se pak systematicky prochází a tím se zjistí kódy pro všechny symboly. Na obr. 1.3 je příklad Huffmanova kódování [1].



symbol	pravděpodobnost	kódová značka
a	0.4	11
b	0,2	00
c	0,15	101
d	0,1	100
e	0,1	011
f	0,05	010

Obr. 1.3. Příklad Huffmanova kódování

1.1.4 Prediktivní metody

Prediktivní metody využívají korelaci mezi sousedními vzorky obrazového signálu. V takovém případě je možné hodnotu vzorku odhadnout z přecházejících vzorků a zakódovat jen chybu, která vznikla při tomto odhadu. Tato chyba následně podléhá modulaci delta, (která se vyznačuje dvoustavovým výstupním kódem) nebo rozdílové pulsně kódové modulaci (DPCM) s více kvantovacími hladinami.

Delta modulace se používá především v komunikační technice při kódování zvuku. Pro kódování obrazů není vhodná kvůli omezení kódování velkých změn signálu a jevu zrnitosti, který se projevuje při malých změnách signálu.

Komprese prediktivní metodou spočívá v tom, že predikcí se omezí dynamika signálu, důsledkem toho je potřeba menšího počtu bitů na zakódování informace. Prediktivní metody komprese obrazových dat dosahují kompresního poměru přibližně 3:1. Nevýhodou prediktivního kódování je malá odolnost vůči kanálovým poruchám. Proto obrazové data po prediktivním kódování podrobena ještě jiným kompresním metodám. Výhodou predikčních metod je jejich jednoduchost.[1]

1.2 Ztrátové kompresní metody

Ztrátové kompresní metody, jinak též redukce irelevance, je v procesu kódování nevratný proces. Proto dochází k informační ztrátě. Množství potlačení irelevantní složky určují vlastnosti subjektu. Vychází se přitom z poznatku, že každý obraz se nějakým způsobem vyhodnocuje a je proto zbytečné přenášet nebo uchovávat informace, které příslušné vyhodnocovací zařízení není schopné zpracovat. Protože ve většině případů je vyhodnocovacím zařízením pozorovatel, redukce informace je založená na zohlednění vlastnosti lidského zraku [1].

Typickou ukázkou nejběžnějších ztrátových kompresních metod je například:

- Kompresní formát JPEG
- Vektorová kvantizace
- Diskrétní kosinová transformace (DCT)

1.2.1 Vektorová kvantizace

Jednou z nejvýkonnějších kompresních metod v současnosti je vektorová kvantizace obrazu. Obraz se nejprve dělí na menší bloky obsahující N=IxJ obrazových prvků. Posloupnost vzorků reálných amplitud jasu obrazu transformujeme na posloupnost vektorů (každý blok chápeme jako N rozměrný vektor). Na vstup kvantizátoru se převádí posloupnost těchto vektorů. Ve vektorovém kvantizátoru se každému takovému vektoru přiřadí jiný vektor, který se získá z paměti, tzv. kódové knihy. Výběr vhodného vektoru z kódové knihy se realizuje na základě podobnosti s přijatým vektorem, popsaný minimální euklidovskou vzdáleností. Když je známa kódová kniha, celý blok obrazu potom postačí popsat kódovým slovem reprezentujícím adresu příslušného vektoru v kódové knize.

Uvedeným způsobem je možné realizovat velmi výhodnou kompresi dat. Efektivní využívání vektorové kvantizace umožňují suboptimálně vektorové kvantizátory, které redukují počet operací při vyhledávání vektorů, v porovnání s optimálními vektorovými kvantizátory.

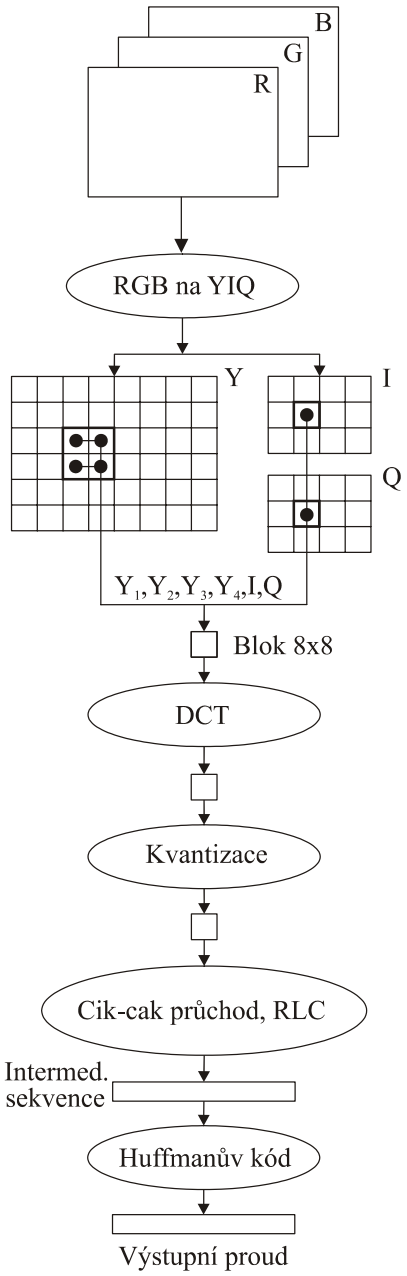
Kvantizátory svými principy funkce vnášejí do procesu zpracování obrazu chyby (tzv. kvantizační). Proto je výběr kvantizátorů omezován požadavky stupně degradace obrazu [1].

1.2.2 Kompresní formát JPEG

Kompresní formát JPEG v sobě zahrnuje více kompresních metod. Nejde tedy pouze o jeden algoritmus ale o celý řetězec algoritmů, jehož výsledkem je nejpopulárnější grafický formát. Na obr. 1.4 je zjednodušené schéma komprese JPEG.

Reprezentace barev v prostoru RGB není z hlediska komprese obrazů příliš výhodná. Lidské oko je citlivější na změny jasu než na změny barvy. K redukci barvonosné informace se barva reprezentuje nejčastěji v některém z prostorů YCbCr, YUV, YIQ. Složka Y je jas, zbývající dvě složky jsou barvonosné.

Komprese JPEG je obvykle prováděna v následujících krocích: Obraz se převede do prostoru umožňující redukci objemu barvonosné informace (např. do prostoru YIQ). Provede se snížení počtu vzorků v barvonosných rovinách. Rovina Y i roviny barvonosné se rozdělí na bloky 8x8 bodů. Další zpracování probíhá po tzv. makroblocích. Makroblok se skládá ze čtyř bloků v rovině Y, kde tyto čtyři bloky vytvářejí oblast 16x16 pixelů, a dále z bloků obsahujících barvonosné složky pro tutéž oblast. Předpokládejme, že hodnoty bloku jsou popsány celými čísly $\langle 0,1,\ldots,2^{P-1}\rangle$. Pro každý blok se provede následující: provede se posunutí hodnot tak, aby byly z intervalu $\langle -2^{P-1},\ldots,2^{P-1}-1\rangle$. Dále se provede diskrétní kosinová transformace bloku (viz. DCT).

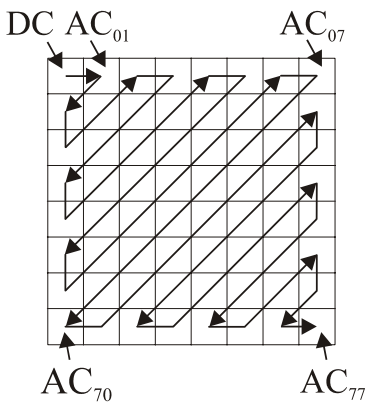


Obr. 1.4. Schéma komprese JPEG

Po provedení kosinové transformace se pro každý zpracovávaný blok provede kvantování. Rozměr kvantovací tabulky je 8x8. Kvantizace se opírá o pozorování, že některé frekvence v obraze (zpravidla vysoké) lze reprezentovat dosti hrubě (tj. na malém počtu bitů), aniž by se obraz pozorovatelně poškodil. Kvantizační tabulka se stává součástí komprimovaného obrazu.

Po provedení kvantizace pokračuje zpracování každého bloku převodem výsledku kvantizace do intermediální "Cik-Cak" sekvence (obr. 1.5). Jeden koeficient se nazývá DC, zbývajících 63 koeficientů jsou označovány jako AC. Tato sekvence se dále komprimuje pomocí RLC. Metoda RLC je velice výhodná vzhledem k charakteru vstupních dat, které obsahují mnoho nul

Poslední fází komprese je zakódovaní intermediální sekvence. Hodnoty jsou zakódovány Huffmanovým kódováním. Připomeňme, že v Huffmanově kódování jsou často používaným symbolům přiřazeny krátké kódy a symbolům používaným méně často kódy delší [4].



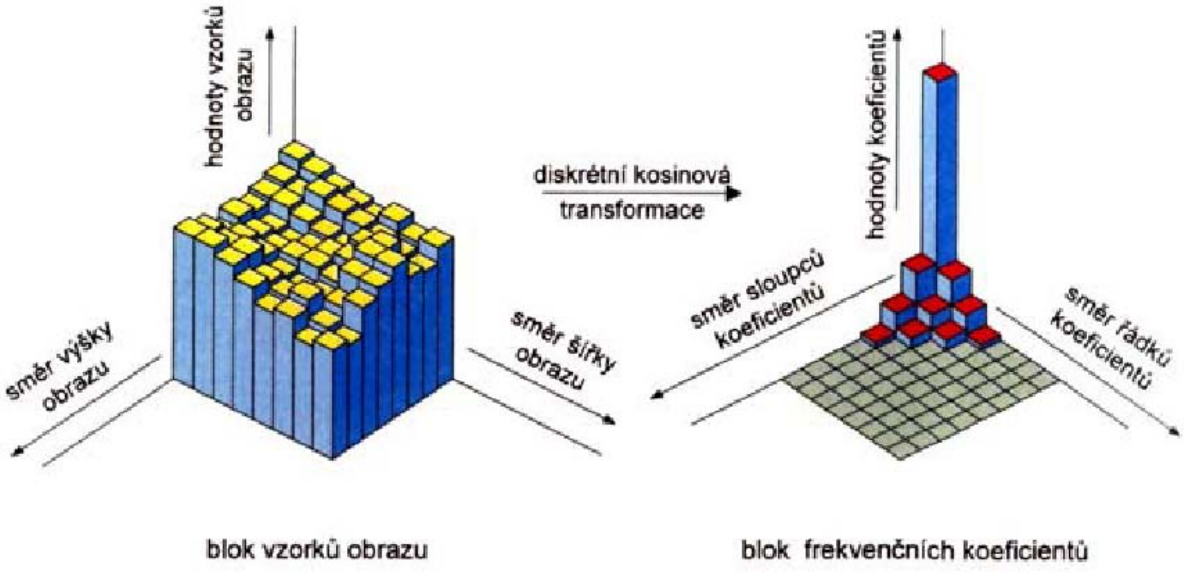
Obr. 1.5. "Cik-cak" sekvence

1.2.3 Diskrétní kosinová transformace (DCT)

DCT patří mezi tzv. transformační metody komprese. Cílem těchto metod je dekorelovat vnitřní vazby a statistické závislosti mezi obrazovými prvky. Tímto způsobem je možné potlačit redundantní složku obrazových dat [1].

DFT poskytuje spektrální koeficienty komplexního charakteru. Je výhodnější využít pouze jejich reálnou nebo imaginární část. Takto je možné principielně přejít z této transformace na diskrétní kosinovou transformaci anebo diskrétní sinovou transformaci.

Kompresní metody založené na DCT jsou ztrátové kompresní metody. Celá 2D DCT komprese spočívá v tom, že kmitočtová reprezentace obrazu je ideálním stavem systému pro potřeby uložení dat – její informační hodnota je značně zhuštěná a velkou většinu koeficientů, které vznikly po transformaci, lze zanedbat. Tímto dosáhneme velmi vysokého stupně komprese. Na obr. 1.6 je ukázka vlivu DCT.



Obr. 1.6. DCT se silně korelovanými vzorky obrazu

2 GIF (Graphics Interchange Format)

2.1 Úvod

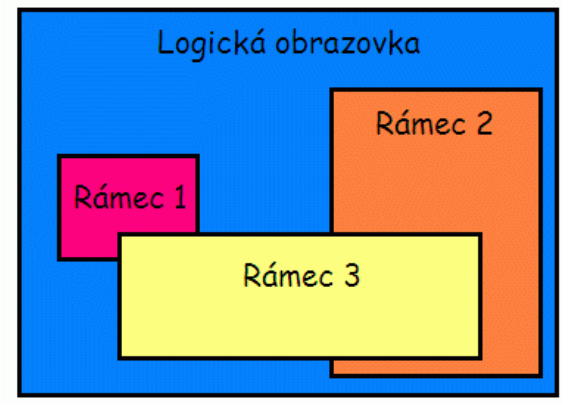
Grafický formát GIF byl vyvinut americkou společností CompuServe. Existuje ve dvou variantách pojmenovaných podle roku svého vzniku. Jedná se o variantu 87a představenou v roce 1987 a variantu 89a z roku 1989. GIF je pravděpodobně nejběžněji používaný grafický formát na Internetu. Je výhodný zejména pro svou malou velikost souboru. Oproti JPEG využívá GIF bezeztrátovou kompresní metodu LZW, což umožňuje vytvořit soubor malé velikosti bez ztráty detailu nebo bez rozmazání obrázku. Hlavní nevýhodou formátu GIF je, že podporuje pouze 256 barev (8 bitů). To znamená, že GIF není určen pro fotografie nebo jiné obrázky, které obsahují veliké množství různých barev. Naopak je vhodný pro zobrazení loga firem, tlačítek, animovaných obrázků, bannerů prostě všech objektů, které používají relativně málo barev a které obsahují velké jednobarevné plochy.

Interní struktura souboru GIF je tvořena tzv. bloky. Každý blok má svou určitou funkci a obsah (viz. dále). Varianta 89a obsahuje v souboru ty samé typy bloků jako její starší předchůdce 87a. Ovšem verze 89a je rozšířena ještě o řadu tzv. rozšiřujících bloků (Extension blocks), např. blok rozšíření grafické kontroly (Graphic Control Extension), bloky obsahující text a komentáře (Comment Extension a Plain Text Extension) nebo aplikační rozšiřující blok (Application Extension). Výše uvedené bloky umožňují efektivnější práci s grafikou obrázku a umožňují tak tvorbu velmi populárních animací nebo vytváření transparentních obrázků atd. [5][6].

2.2 Obrázky a rámce

GIF je určen především pro záznam a pozdější přenos grafických informací uložených ve formě bitmapy. Grafické formáty, jako například JPEG, PNG či BMP obsahují popis rastrového obrázku jako celku, přičemž se ukládají barvové hodnoty všech pixelů, ze kterých se rastrový obrázek skládá.

V grafickém formátu GIF je tomu jinak. Celý obrázek není v souboru uložen jako jeden celek, ale skládá se z několika rámců (frames), což jsou vlastně obdélníkové oblasti umístěné uvnitř logické obrazovky (Logical Screen). Minimálně musí být vždy přítomen jeden rámec, jejich maximální množství však není omezeno. Každý rámec je možné chápat jako rastrový obrázek, který je celou svou plochou umístěn v logické obrazovce – podle specifikace nesmí žádný pixel z rámce ležet mimo logickou obrazovku. Pozice rámce v logické obrazovce je určena souřadnicí jeho horního levého rohu a velikostí (šířkou, výškou) zadanou v pixelech. Velmi důležitou součástí popisu logické obrazovky je index jedné barvy v globální barvové paletě, která bude použita pro vykreslení pozadí.

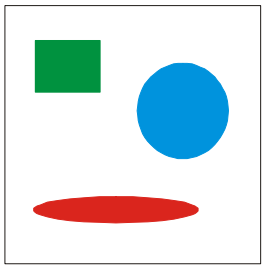


Obr. 2.1. Ukázka umístění rámců uvnitř logické obrazovky

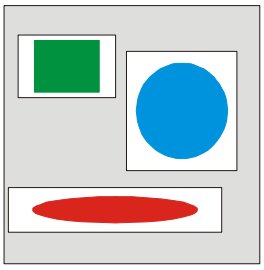
Na obrázku 2.1 je naznačen možný způsob, jak mohou být rámce umístěny v logické obrazovce. Je vidět, že není nutné,

aby rámce pokryly celou logickou obrazovku. Rámce se dokonce mohou navzájem překrývat. Toho je velmi intenzívně využíváno při tvorbě animovaných obrázků.

Rámce nachází v grafickém formátu GIF mnoho uplatnění. Na obr. 2.2 je obrázek, který je uložený pouze v jednom rámci. Oproti tomu na obr. 2.3 je tentýž obrázek, který je uložen ve třech rámcích. Rozdíl je patrný. Zatímco obrázek uložený v jednom rámci je nutný ukládat i s prázdnou bílou plochou, tak u obrázku uloženého ve třech rámcích je zapotřebí uložit jen data příslušící jednotlivým rámcům. Bílá plocha mimo rámce (v obr. 2.3 je pro ilustraci změněna na šedou barvu) se pak vykreslí jako barva pozadí. Tím se dosahuje redukce celkové velikosti obrázku [6][5].



Obr. 2.2. Obrázek uložený v jednom rámci



Obr. 2.3. Obrázek uložený ve třech rámcích

2.3 Barvová paleta

Každý pixel v obrázku GIF může být popsán maximálně osmi bity. To znamená, že jeho barvu je možné vybrat z barvové palety obsahující nejvýše 256 barev. Každá barva je v barvové paletě popsána trojicí hodnot (triplet) – barvových složek označených písmeny R (red), G (green) a B (blue), přičemž každá barvová složka je uložena v jednom bytu. Maximální velikost barvové palety s 256 barvami je tedy 768 bytů (256×3). U mnoha souborů obrázků se proto může stát, že barvová paleta tvoří nezanedbatelnou část jejich celkové velikosti.

V grafickém formátu GIF rozlišujeme dvě základní barvové palety: globální a lokální. **Globální barvová paleta** (Global Colour Table) se v souboru může objevovat maximálně jednou. Její velikost je udána v bloku *popis logické obrazovky*. Pokud je globální barvová paleta aktivní, tak všechny rámce, ze kterých je obrázek složen, mohou tuto barvovou paletu využít. V některých případech není globální paleta barev přítomna. V tom okamžiku musí mít každý rámec, který obrázek obsahuje, zadanou svoji **lokální barvovou paletu** (Local Colour Table). Velikost lokální barvové palety je určena v hlavičce konkrétního rámce.

V případě, že je v obrázku přítomna globální i lokální barvová paleta, dekodér musí pro rámec, který má definovanou lokální barvovou paletou, použít právě tuto lokální barvovou paletu místo globální barvové palety. Doporučuje se, aby kodér používal pouze globální barvovou paletu, pokud všechny rámce obrázku mohou být pomoci této palety zobrazeny. V případě, že není přítomna ani globální ani lokální barvová paleta, měl by prohlížecí program použít systémovou paletu [5].

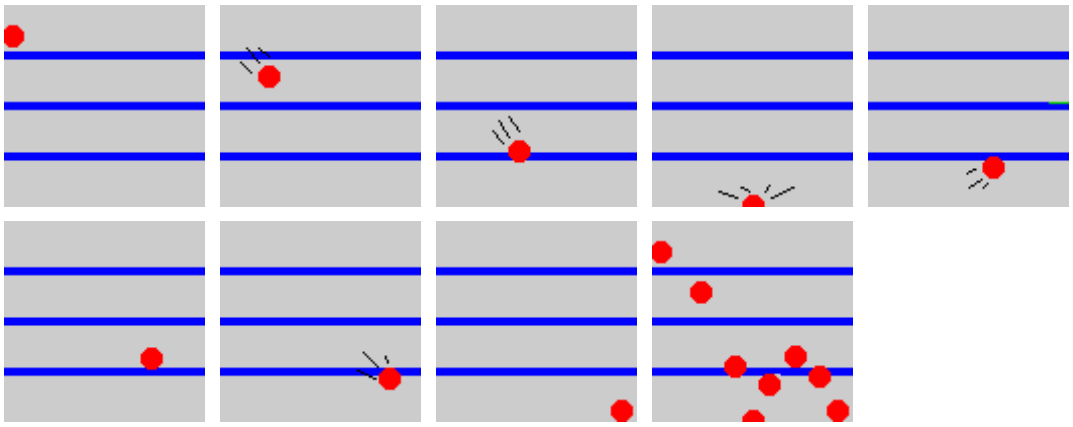
2.4 Transparentní pixely

Verze GIF89a přinesla oproti původní verzi GIF87a jednu podstatnou novinku. Pomocí bloku *rozšíření grafické kontroly* (Graphics Control Extension) je možné v každém rámci specifikovat jeden index do aktivní barvové palety, který je vyhrazen pro průhledné pixely.

Díky tomu je tedy možné, aby optický tvar rámce nebyl pouze obdélníkový, ale v podstatě libovolného tvaru. Vzhledem k tomu, že je možné transparentními pixely překreslit i původně neprůhlednou barvu pozadí, může mít i obrázek jako celek neobdélníkový tvar, což je velmi často využíváno zejména při tvorbě různých log na webových stránkách a v neposlední řadě i v řadě animací [6].

2.5 Podpora animací

Grafický formát GIF je na Internetu oblíbený zejména z toho důvodu, že podporuje animace. Ve skutečnosti nejsou animace nic jiného, než sled po sobě jdoucích rámců (které mohou ležet na sobě), mezi jejichž zobrazením je vložena krátká či delší časová prodleva. Tuto prodlevu lze nastavit každému rámci pomoci bloku *rozšíření grafické kontroly* (viz. 2.7.9.). Časová délka těchto prodlev může být různá. Pokud jsou rámce překreslovány rychlostí zhruba 10 snímků za sekundu, vzniká dojem poměrně plynulé animace. Animace byly zavedeny až ve verzi GIF89a [6]. Na obr. 2.4 je rozfázovaný animovaný GIF složený z osmi rámců. Poslední obrázek je pak součet předchozích osmi rámců.



Obr. 2.4. Rozfázovaný animovaný GIF

2.6 Prokládání

Již ve verzi GIF87a byla zavedena možnost zajímavého a mnohdy velmi užitečného způsobu ukládání obrazových řádků v jednotlivých rámcích. Jedná se o takzvané prokládání (interlacing). Prokládání funguje tím způsobem, že jednotlivé obrazové řádky nejsou v rámci uloženy ve své přirozené sekvenci (1, 2, 3 ... n), ale hierarchicky podle schématu, při jehož použití se obrázky vykreslují ve čtyřech průchodech, jak je naznačeno v tabulce 2.1.

Tabulka 2.1. Princip prokládání(převzato z [6])

Řádek	1. průchod	2. průchod	3. průchod	4. průchod	Výsledek
0	**1a**				**1a**
1				**4a**	**4a**
2			**3a**		**3a**
3				**4b**	**4b**
4		**2a**			**2a**
5				**4c**	**4c**
6			**3b**		**3b**
7				**4d**	**4d**
8	**1b**				**1b**
9				**4e**	**4e**
10			**3c**		**3c**
11				**4f**	**4f**
12		**2b**			**2b**

Nejprve jsou tedy v souboru uloženy řádky 0, 8, 16, 24 atd., tj. jedna osmina celého rámce. Následuje druhý průchod, ve kterém jsou uloženy řádky 4, 12, 20..., tj. druhá osmina rámce. Ve třetím průchodu je již uložena celá čtvrtina řádků, a to řádky 2, 6, 10, 14 (každý čtvrtý řádek). Zbývající obrazové řádky jsou uloženy ve čtvrtém a současně i posledním průchodu.

Při přenosu obrázků ze serveru ke klientovi může uživatel získat představu o celém obrázku již po přenesení jedné osminy dat, tj. po prvním průchodu a přenos popř. zastavit. Na obr. 2.5 je příklad prokládaného a neprokládaného řádkování při načtení poloviny obrázku. Jedinou nevýhodou prokládaných obrázků typu GIF je poněkud větší velikost celého souboru, neboť se sníží vzájemná podobnost pixelů mezi jednotlivými řádky, což zapříčiní horší účinnost algoritmu LZW.

Obr. 2.5. Příklad neprokládaného řádkování (vlevo) a prokládaného řádkování (vpravo) při načtení poloviny obrázku (převzato z [7]).

2.7 Interní struktura grafického formátu GIF

Pro lepší pochopení interní struktury souboru typu GIF je zapotřebí vysvětlit, jakým způsobem jsou interní data souboru vytvářena.

Kodér je program (algoritmus), který na základě rastrových dat a dalších informací produkuje určité nezbytné kontrolní a datové bloky, které jsou velmi důležité pro správnou reprodukci originálního obrázku.

Kodér musí mít tyto základní vlastnosti:

- do výstupních dat musí začlenit data (bloky) pro reprodukci obrázku.
- musí zajistit, aby výstupní data nesla informaci o nejstarší možné verzi souboru, která definuje, jaké bloky se v souboru nachází. Je to snaha zajistit, aby co nejvíce dekodérů mohlo tyto data zpracovávat.
- zajistit takový způsob kódovaní dat, aby byl dekódovací proces optimalizován. Kodér musí co nejvíce potlačit redundantní informace v datech.
- v rámci možností se vyhnout skupinám dat, které vyžadují vynulování hardwarových parametrů v průběhu dekódovacího procesu.
- nastavit na nulu všechny bity, které jsou určeny pro rezervování.

Na výstupu kodéru jsou tedy zakódovaná data (bloky), která tvoří interní data grafického formátu typu GIF. Aby bylo možné takto zakódovaný obrázek zobrazit, je zapotřebí dekodér.

Dekodér je program (algoritmus), který správným způsobem dekóduje interní data souboru. Dekodér tyto data zpracovává sekvenčně. Postupně tedy rozebírá jednotlivé bloky a sub-bloky pomocí kontrolních informací k nastavení hardwaru a pomocí parametrů sloužících k samotnému zpracování. Dále interpretuje data k vykreslení obrázku.
Dekodér musí mít tyto základní vlastnosti:

- sekvenčně dekódovat každý rámec. Nesmí dekódovat rámce s jiným zpožděním, než jaké je nastaveno u každého rámce.
- při kódování obrázku jsou do výstupního souboru začleněny informace v podobě hardwarových parametrů. Tyto parametry informují dekodér o tom, jak má tyto hardwarové parametry nastavit pro úspěšné dekódování. Dekodér by měl tyto parametry nastavit co nejpřesněji.

Interní strukturu souboru GIF lze tedy chápat jako sekvenci bytů. Počet těchto bytů udává zároveň i velikost souboru. Všechny tyto byty lze klasifikovat do určitého množství bloků, které tvoří celý obsah souboru. Bloky dělíme do tří skupin:

1. Kontrolní bloky (Control blocks)
    - Hlavička (Header)
    - Popis logické obrazovky (Logical Screen Descriptor)
    - Rozšíření grafické kontroly (Graphic Control Extension)
    - Ukončující znak souboru (Trailer)

2. Bloky pro interpretaci obrázku (Graphic-Rendering Blocks)
    - Popis rámce (Image Descriptor)

- Rozšíření o jednoduchý text (Plain Text Extension)

3. Bloky pro zvláštní účel (Special Purpose Blocks)
   - Aplikační rozšíření (Application Extension)
   - Rozšíření o komentář (Comment Extension)

Kontrolní bloky obsahují informace pro nastavení parametrů hardwaru, také slouží ke kontrole zpracování dat.

Bloky pro interpretaci obrázku obsahují informace a data sloužící pro zobrazení obrázku na zobrazovacím zařízení (monitor).

Bloky pro zvláštní účel jsou transparentní vůči dekódovacímu procesu.

Některé výše uvedené bloky jsou povinné, některé se musí vyskytovat na určitém místě souboru, některé se mohou opakovat apod. V tabulce 2.2 je přehledně zobrazeno, které bloky se musí resp. nemusí vyskytovat v souboru, jestli je možné tyto bloky v rámci jednoho souboru opakovat. Dále je u každého bloku uvedeno, zda se jeho definice vyskytovala už ve verzi GIF87a či až v novější verzi GIF89a [5]

Tabulka 2.2. Přehled jednotlivých bloků (převzato z [6])

Název bloku	Povinná položka?	Lze opakovat?	Verze
Hlavička	Ano	Ne	87a
Popis logické obrazovky	Ano	Ne	87a
Globální barvová paleta	Ano/Ne	Ne	87a
Rozšíření grafické kontroly	Ne	Ano	89a
Popis rámce	Ano	Ano	87a
Lokální barvová paleta	Ano/Ne	Ano	87a
Rozšíření o komentář	Ne	Ano	89a
Rozšíření o jednoduchý text	Ne	Ano	89a
Aplikační rozšíření	Ne	Ano	89a
Ukončovací znak GIF souboru	Ano	Ne	87a

2.7.1 Sub-bloky (Data Sub-blocks)

Sub-bloky obsahují data a nemají žádné označení. Mohou obsahovat jakákoli data (byty), např. data rámce. První byte sub-bloku udává velikost sub-bloku v bytech. Velikost sub-bloku je od 0 bytů do 255 bytů. Prázdný sub-blok je blok, jehož první byte udávající velikost sub-bloku obsahuje hodnotu 0x00 [5].

2.7.2 Ukončující znak bloku (Block Terminator)

Tento blok má velikost jeden byte a hodnota tohoto bytu je 0x00. Slouží jako ukončovací znak např. sub-blokům s daty nebo rozšiřujícím blokům [5].

2.7.3 Hlavička (Header)

Hlavička je povinná a pro každý soubor se musí vyskytovat pouze jednou a to hned na začátku souboru. Hlavička má velikost šest bytů a její struktura odpovídá obr. 2.6.

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Signatura							
	2								
	3								
	4	Verze souboru							
	5								
	6								

Obr. 2.6. Vnitřní struktura bloku

Signatura (Byty 1-3) – identifikuje data v souboru jako GIF a zároveň označuje začátek dat souboru typu GIF. V ASCII kódu má toto pole hodnotu 'GIF'.

Verze souboru (Byty 4-6) – číslo verze souboru identifikuje minimální nezbytné požadavky pro dekodér k úplnému zpracování všech dat v souboru. V ASCII kódu má toto pole hodnotu např. '89a' [5].

2.7.4 Popis logické obrazovky (Logical Screen Descriptor)

Na obr. 2.7 je struktura bloku popisující logickou obrazovku. Obsahuje parametry nezbytné k definování oblasti zobrazovacího zařízení, uvnitř které se má obrázek zobrazit. Pozice této oblasti se vztahuje k levému hornímu rohu virtuální obrazovky. Tento blok je povinný a každý soubor ho obsahuje maximálně jednou. Musí následovat ihned po hlavičce [5].

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Šířka logické obrazovky							
	2								
	3	Výška logické obrazovky							
	4								
	5	a	b			c	d		
	6	Index barvy pozadí							
	7	Pixel Aspect Ratio							

Obr. 2.7. Vnitřní struktura bloku

Šířka a výška logické obrazovky (Byty 1-4) – je udávána v pixelech, určuje šířku a výšku logické obrazovky, kde se jednotlivé rámce zobrazují.

Bitové pole (Byte 5)

a) Použití globální barvové palety – je-li tento bit (MSB) nastaven na hodnotu 1, pak je v souboru použita globální barvová paleta. V tom případě je i použit index barvy pozadí (byte 6) v globální barvové paletě pro barvu pozadí. Blok s globální barvovou paletou následuje hned po bloku popisující logickou obrazovku.

b) Barvové rozlišení – hodnota daná těmito třemi bity zvýšená o jedničku udává počet bitů každé základní barvy (RGB).

c) Setřídění – bit indikuje, zda je globální barvová paleta setříděna. Má-li hodnotu 1, pak je paleta setříděna sestupně podle důležitosti každé barvy. Má-li hodnotu 0, není paleta setříděna vůbec.

d) Velikost globální barvové palety – jestliže je v obrázku použita globální barvová paleta, tak hodnota daná těmito třemi bity se používá k výpočtu velikosti globální barvové palety v bytech podle vztahu:

$$Velikost\_palety = 3 \cdot 2^{(hodnota\_tříd\_bitů+1)} \quad \text{[Bytů]}. \qquad (2.1)$$

Index Barvy pozadí (Byte 6) – index v globální barvové paletě pro barvu pozadí. Barva pozadí slouží k vykreslení těch pixelů, které nejsou překryty jednotlivými rámci. Jestliže není použita globální barvová paleta, bude hodnota tohoto bytu rovna nule a tento byte bude ignorován.

Pixel Aspect Ratio (Byte 7) - obsahuje číslo rozlišovacího poměru šířky ku výšce pixelu v obrázku. Jestliže hodnota tohoto pole není 0x00, pak je jeho hodnota použita pro výpočet poměru:

$$Aspect\_Ratio = \frac{(Pixel\_Aspect\_Ratio)+15}{64} \quad [-]. \qquad (2.2)$$

2.7.5 Globální barvová paleta (Global Colour Table)

Blok globální barvové palety je sekvence bytů reprezentující RGB triplety. Struktura bloku odpovídá obr. 2.8. Je vidět, že velikost globální barvové palety je maximálně 768 bytů. Velikost palety je rovna dle vztahu 2.1. Globální barvovou paletu používají rámce bez definované lokální barvové palety a blok rozšíření o jednoduchý text. Její přítomnost v souboru je indikována v bloku popisující logickou obrazovku [5].

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Červená index 0							
	2	Zelená index 0							
	3	Modrá index 0							
	4	Červená index 1							
	...	……..							
	...	……..							
	767	Zelená index 255							
	768	Modrá index 255							

Obr. 2.8. Vnitřní struktura bloku

2.7.6 Popis rámce (Image Descriptor)

Každý obrázek GIF musí obsahovat minimálně jeden rámec a každý rámec musí mít blok popisující tento rámec. Rámce musí ležet uvnitř logické obrazovky celým svým povrchem. Blok popisující rámec obsahuje parametry nezbytné ke zpracování dat rámce. Tento blok patří do skupiny bloků sloužících pro interpretaci obrázku. Mohou mu tedy předcházet jeden nebo více kontrolních bloků jako třeba blok rozšíření grafické kontroly. Struktura bloku je na obr. 2.9 [5].

Oddělující znak rámců (Byte 1) - identifikuje začátek bloku popisující rámec. Hodnota tohoto bytu je fixně dána 0x2C.

Pozice rámce (Byty 2-5) – tyto čtyři byty udávají pozici (v pixelech) levého horního rohu rámce v logické obrazovce.

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Oddělující znak rámců							
	2	Pozice horního okraje rámce							
	3								
	4	Pozice levého okraje rámce							
	5								
	6	Šířka rámce							
	7								
	8	Výška rámce							
	9								
	10	a	b	c	d		e		

Obr. 2.9. Vnitřní struktura bloku

Šířka a výška rámce (Byty 6-9) – udává výšku a šířku rámce v pixelech.

Bitové pole (Byte 10)

a) Použití lokální barvové palety - je-li tento bit (MSB) nastaven, pak je pro tento rámec definována jeho lokální barvová paleta, která následuje bezprostředně za blokem popisující rámec. Není-li lokální barvová paleta použita, pak bezprostředně za tímto blokem následují data rámce.

b) Prokládání – bit indikuje, jsou-li data rámce prokládán nebo nikoli. Je-li bit nastaven na 1, je použito prokládání.

c) Třídění – bit indikuje, zda je lokální barvová paleta setříděna. Je-li tento bit nastaven na 1, pak je lokální barvová paleta setříděna podle klesající důležitosti každé barvy.

d) Rezervované bity

e) Velikost lokální barvové palety – jestliže je v rámci použita lokální barvová paleta, tak hodnota daná těmito třemi bity se používá k výpočtu velikosti lokální barvové palety v bytech podle vztahu (2.1).

2.7.7 Lokální barvová paleta (Local Colour Table)

Blok obsahující lokální barvovou paletu je sekvence bytů reprezentující RGB triplety. Tuto paletu využívá rámec, jehož data se nachází bezprostředně za ní. Její přítomnost je indikována v bloku popis rámce a její velikost je dána třemi bity v posledním bytu (byte 10) v bloku popis rámce. Velikost palety je rovna dle vztahu 2.1. a její velikost může být maximálně 768 bytů. Struktura bloku je na obr. 2.10.

Jestliže je tedy lokální barvová paleta pro rámec použita, tak tato paleta se stává dočasně aktivní a příslušný rámec bude využívat její barvy. Z toho plyne, že blok s lokální barvovou paletou je ryze volitelný [5].

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Červená index 0							
	2	Zelená index 0							
	3	Modrá index 0							
	4	Červená index 1							
	...	…….							
	...	…….							
	767	Zelená index 255							
	768	Modrá index 255							

Obr. 2.10. Vnitřní struktura bloku

2.7.8 Data rámce (Image Data)

Na obr. 2.11 je znázorněna struktura bloku, který obsahuje rastrová data, jenž přísluší konkrétnímu rámci. Skládá se ze sekvence sub-bloků o maximální velikost 255 bytů. Těchto sub-bloků může za sebou následovat teoreticky nekonečné množství. Tyto sub-bloky obsahují zakódované indexy do aktivní palety barev. Indexy pixelů rámce jsou brány v pořadí zleva doprava, od shora dolů. Každý index musí být obsažen v aktivní paletě barev. Sekvence indexů je kódována pomoci LZW algoritmu s proměnnou délkou kódu [5].

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Minimální velikost pro LZW kód							
	2..N	Sub - bloky s daty							

Obr. 2.11. Vnitřní struktura bloku

Minimální velikost pro LZW kód (Byte 1) – hodnota tohoto bytu stanoví inicializační počet bitů určených pro LZW kód.

Sub – bloky s daty (Byte 2 - N) – sub–bloky se zakódovanými indexy do aktivní palety barev.

2.7.9 Rozšíření grafické kontroly (Graphic Control Extension)

Tento blok obsahuje parametry, které modifikují blok popis rámec a blok rozšíření o jednoduchý text. Také těmto blokům zpravidla předchází. Jeho struktura je na obr. 2.12. Obsahuje informace o tom, jak interpretovat grafickou informaci, např. jestli bude v průběhu vykreslování obrázku nějaké časové zpoždění. Rozšíření grafické kontroly je volitelný blok a podporuje ho pouze verze 89a [5].

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Uvaděč rozšiřujícího bloku							
	2	Identifikátor							
	3	Velikost bloku							
	4	a			b			c	d
	5	Doba Zpoždění							
	6								
	7	Index transparentní barvy							
	8	Ukončující znak bloku							

Obr. 2.12. Vnitřní struktura bloku

Uvaděč rozšiřujícího bloku (Byte 1) – identifikuje začátek rozšiřujícího bloku. Hodnota tohoto bytu je fixně dána 0x21.

Identifikátor (Byte 2) – identifikuje tento blok jako blok *rozšíření grafické kontroly*. Hodnota tohoto bytu je fixně dána 0xF9.

Velikost bloku (Byte 3) – následující počet bytů bloku s fixní hodnotou 0x04.

Bitové pole (Byte 4)

a) Rezervované bity

b) Volná dispoziční metoda – definuje možnosti, jak bude s obrázkem nakládáno po jeho vykreslení. Nabývá pěti možných hodnot

0 - Žádná dispoziční metoda. Dekodér neprovádí žádnou akci.
1 - Zákaz dispozice. Obrázek má být ponechán na svém místě.
2 - Obrázek musí mít takovou barvu pozadí, jako je barva oblasti, kde se vykresluje.
3 - Obnovit původní oblast. Dekodér musí obnovit oblast, na které se obrázek vykresluje do své původní podoby.
4-7 - Tyto hodnoty jsou určeny k volnému definování.

c) Uživatelský vstup – indikuje, zda je nebo není očekáván vnější zásah uživatele před pokračováním. Je-li tento bit nastaven na 1, pak zpracovávání (dekódování) dat bude pokračovat až po zásahu uživatele. Tímto zásahem může být např. kliknutí myší atd.

d) Transparentnost - bit (LSB) indikuje, zda je použit index pro transparentní barvu, který je deklarován v sedmém bytu. Je-li hodnota toho bitu 1, pak je transparentní barva použita a její index v barvové paletě odpovídá hodnotě v sedmém bytu.

Doba zpoždění (Byte 5-6) – jestliže hodnoty těchto dvou bytů nejsou 0, pak jejich hodnota udává počet setin vteřin, které musí dekodér vyčkat před pokračováním v dekódování dat obrázku. Čas začíná ubíhat ihned jakmile se celý obrázek (rámec) přeloží.

Index transparentní barvy (Byte 7) – bit pro aktivní transparentnost musí být nastaven na hodnotu 1. Jakmile dekodér při dekódování narazí na pixel, který je definován jako transparentní, tak se tento pixel nezpracovává a dekódování automaticky přechází na následující pixel.

Ukončující znak bloku (Byte 8) – ukončuje rozšíření grafické kontroly (0x00).

2.7.10 Rozšíření o komentář (Comment Extension)

Rozšíření o komentář obsahuje textové informace, které se v obrázku nezobrazují. Struktura bloku je uvedena obr. 2.13. Blok je vhodný pro komentáře týkající se grafiky, textové příspěvky nebo různé popisky atd. Může se vyskytovat na jakémkoli místě v souboru. Nicméně se doporučuje aby neinterferoval s kontrolními bloky nebo s daty. Měl by být umístěn na začátku nebo na konci souboru. Rozšíření o komentář je volitelný blok a podporuje ho pouze verze 89a [5].

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Uvaděč rozšiřujícího bloku							
	2	Identifikátor							
	N	Data komentáře							
	N+3	Ukončující znak bloku							

Obr. 2.13. Vnitřní struktura bloku

Uvaděč rozšiřujícího bloku (Byte 1) – identifikuje začátek rozšiřujícího bloku. Hodnota tohoto bytu je fixně dána 0x21.

Identifikátor (Byte 2) – identifikuje tento blok jako *rozšíření o komentář*. Hodnota tohoto bytu je fixně dána 0xFE.

Data komentáře – sekvence sub-bloků, každý o maximální velikosti 255 bytů. Těchto sub-bloků může být za sebou několik. Celá sekvence sub-bloků je pak ukončena ukončujícím znakem s hodnotou 0x00. Tento blok je určen pro uživatele, obsahuje text využívající 7-bitových ASCII znaků.

2.7.11 Rozšíření o jednoduchý text (Plain Text Extension)

Na obr. 2.14 je struktura bloku rozšíření o jednoduchý text. Obsahuje textová data a důležité parametry sloužící k zobrazení těchto textových dat jako grafiky, která je součástí zobrazeného obrázku. Tento text je kódován jako 7-bitové tisknutelné ASCII znaky. Textová

data jsou zobrazena pomoci speciální mřížky. Vlastnosti mřížky jsou definovány v bloku. Každý znak je vykreslen v individuální buňce této mřížky a jeho velikost a font je zvolen tak, aby co nejlépe vyhovoval velikosti buňky. Data představující znaky jsou brána sekvenčně a jsou zobrazována uvnitř buněk. Směr zápisu do mřížky je od levé buňky doprava a shora dolů. Textová data jsou zobrazena jakmile jsou všechna data zapsána do tabulky a tabulka je plná. Tento blok vyžaduje, aby v souboru byla přítomná globální barvová paleta, ze které čerpá barvy pro vykreslení barev textu. Rozšíření o jednoduchý text patří do skupiny bloků pro interpretaci grafiky a může být tedy modifikován blokem rozšíření grafické kontroly. Tento blok je volitelný a podporuje ho pouze verze 89a [5].

Uvaděč rozšiřujícího bloku (Byte 1) – Identifikuje začátek rozšiřujícího bloku. Hodnota tohoto bytu je fixně dána 0x21.

Identifikátor (Byte 2) – Identifikuje tento blok jako rozšíření o jednoduchý text. Hodnota tohoto bytu je fixně dána 0x01.

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Uvaděč rozšiřujícího bloku							
	2	Identifikátor							
	3	Velikost bloku							
	4	Pozice levého okraje mřížky							
	5								
	6	Pozice horního okraje mřížky							
	7								
	8	Šířka mřížky							
	9								
	10	Výška mřížky							
	11								
	12	Šířka buňky							
	13	Výška buňky							
	14	Index barvy popředí textu							
	15	Index barvy pozadí textu							

		Bity							
		7	6	5	4	3	2	1	0
Byty	N	Textová data							
	N+1	Ukončující znak bloku							

Obr. 2.14. Vnitřní struktura bloku

Velikost bloku (Byte 3) – počet bytů bloku s fixní hodnotou 0x0C.

Pozice levého okraje mřížky (Byte 4-5) – pozice levého okraje mřížky v pixelech s ohledem na levý okraj logické obrazovky.

Pozice horního okraje mřížky (Byte 6-7) – pozice horního okraje mřížky v pixelech s ohledem na horní okraj logické obrazovky.

Šířka mřížky (Byte 8-9) – šířka mřížky v pixelech.

Výška mřížky (Byte 10-11) – výška mřížky v pixelech.

Šířka buňky (Byte 12) – šířka buňky mřížky udávaná v pixelech.

Výška buňky (Byte 13) – výška buňky mřížky udávaná v pixelech.

Index barvy popředí textu (Byte 14) – index v globální barvové paletě sloužící k zobrazení barvy popředí daného textu.

Index barvy pozadí textu (Byte 15) – index v globální barvové paletě sloužící k zobrazení barvy pozadí daného textu.

Textová data – sekvence sub-bloků, zakončená ukončujícím znakem bloku (0x00).

2.7.12 Aplikační rozšíření (Application Extension)

Obsahuje aplikační a specifické informace. Struktura bloku je na obr. 2.15. Tento blok je volitelný a obsahuje ho pouze verze 89a [5].

		Bity							
		7	6	5	4	3	2	1	0
Byty	1	Uvaděč rozšiřujícího bloku							
	2	Identifikátor							
	3	Velikost bloku							
	4	Aplikační identifikátor							
	…								
	11								
	12	Aplikační ověřující kód							
	13								
	14								

		Bity							
		7	6	5	4	3	2	1	0
Byty	N	Aplikační data							
	N+1	Ukončovací znak bloku							

Obr. 2.15. Vnitřní struktura bloku

Uvaděč rozšiřujícího bloku (Byte 1) – identifikuje začátek rozšiřujícího bloku. Hodnota tohoto bytu je fixně dána 0x21.

Identifikátor (Byte 2) – identifikuje tento blok jako *aplikační rozšiřující blok*. Hodnota tohoto bytu je fixně dána 0xFF.

Velikost bloku (Byte 3) – následující počet bytů bloku s fixní hodnotou 0x0B.

Aplikační identifikátor (Byte 4-11) – sekvence osmi tisknutelných ASCII znaků určených pro identifikaci aplikace, která toto rozšíření využívá.

Aplikační ověřující kód (Byte 12-14) – sekvence tří bytů určených pro ověření aplikačního identifikátoru. Aplikační program by měl použít algoritmus pro výpočet binárního kódu, který jedinečně identifikuje aplikaci, která tohoto aplikačního rozšíření využívá.

Aplikační data – sekvence sub-bloků, zakončená ukončujícím znakem bloku (0x00).

2.7.13 Ukončující znak souboru (Trailer)

Tento blok je povinný a má velikost jednoho bytu. Indikuje konec souboru GIF. Hodnota tohoto bytu je 0x3B neboli znak ";" v ASCII kódu [5].

Na obr. 2.16 je ukázka interní struktury konkrétního souboru. Obrázek má velikost 35B a rozměr 1x1 pixel. Čísla nad tabulkou udávají pořadí bytů a čísla v jednotlivých buňkách udávají hodnoty bytů v hexadecimální soustavě.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
47	49	46	38	39	61	01	00	01	00	80	00	00	FF	80	00	FF	FF	FF
Hlavička						Popis logické obrazovky							Globální barvová paleta					

19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34
2C	00	00	00	00	01	00	01	00	00	02	02	44	01	00	3B
Popis rámce										Data rámce					;

Obr. 2.16. Ukázka interní struktury konkrétního souboru

3 Aplikace pro zobrazení interní struktury souboru typu GIF

Na základě získaných předchozích teoretických znalostí lze navrhnout a realizovat výukovou aplikaci pro přehledné zobrazení interní struktury vybraného souboru typu GIF. Celá aplikace je zamýšlena pro podporu výuky v oblasti počítačové grafiky. Při návrhu byly stanoveny následující požadavky na funkci celé aplikace:

1. Načtení a zobrazeni libovolného (i animovaného) obrázku typu GIF
2. Zobrazení jeho základních vlastností jako např. bitová hloubka, šířka, výška atd.
3. Výpis interních dat vybraného souboru v hexadecimální soustavě do tabulky
4. Výběr libovolného typu bloku, zobrazeni informací o jeho pozici v rámci souboru, detailní zobrazení jeho obsahu a vyznačení vybraného typu bloku v tabulce
5. Umožnění základních operací s obrázky (změna dat, vymazání dat, export/import dat mezi soubory)

Jako vývojové prostředí byl zvolen program MATLAB GUI, který umožňuje velmi efektivní tvorbu různých aplikací s využitím grafického rozhraní. Mimo výše uvedených požadovaných funkcí je velký důraz kladen na jednoduchost a přehlednost celé aplikace. Z tohoto požadavku vyplývá potřeba návrhu vhodného rozhraní s ideálním rozmístěním použitých komponent.

Návrh aplikace lze tedy rozdělit na dvě části: první část zahrnuje tvorbu algoritmů pro získání potřebných informací o všech typech bloků, které vybraný soubor GIF obsahuje. Druhou část tvoří návrh grafického rozhraní pro efektivní zobrazení získaných informací o jednotlivých blocích. Je nutné však podotknout, že celý vývoj aplikace, tedy jak tvorba algoritmů, tak realizace grafického rozhraní, vzniká paralelně. Tedy každá výše zmíněná funkce aplikace je nejprve realizována na bázi algoritmů ovšem s ohledem na funkční možnosti použité komponenty v grafickém rozhraní.

Obr. 3.1. Zjednodušený vývojový diagram

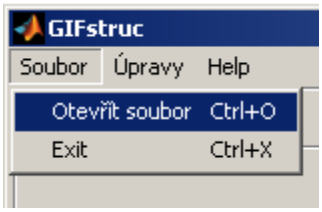
3.1 Realizace algoritmů programu

Na obr. 3.1 je zjednodušený vývojový diagram celé aplikace, který se skládá ze tří částí. V první části je nutné efektivně načíst obsah vybraného souboru a jeho interní hodnoty převést do použitelnější formy. V druhé části, která je nejdůležitější, jsou na základě algoritmů získány potřebně informace o blocích. Poslední fází je efektivní zobrazení získaných informací o jednotlivých blocích do nejvhodnějších komponent v grafickém rozhraní.

3.1.1 Načtení souboru

Vzhledem k tomu, že vnitřní obsah souboru typu GIF tvoří různé znaky, je prvotním úkolem převést tyto znaky do srozumitelnější formy a uložit je do vhodné proměnné k následnému zpracování. Nejefektivnější způsob je načtení souboru pomoci funkce *fread*. Následující jednoduchý zdrojový kód představuje načtení obsahu souboru a jeho převod do dekadické resp. hexadecimální formy.

```
fp = fopen(var,'r');
Read_Dec = fread(fp, 'uint8');
cl = fclose('all');
Hex_obsah = dec2hex(Read_Dec);
```



Obr. 3.2. Menu pro výběr obrázku

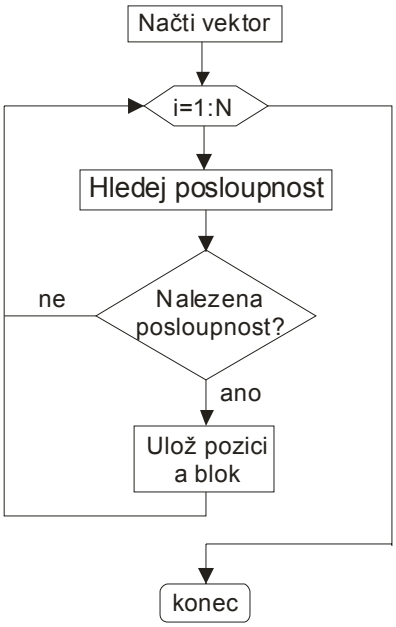
Funkce `fopen` otevře vybraný soubor pro čtení (parametr `'r'`), jehož pozice v adresářové struktuře na pevném disku je dána proměnnou `var` (např. C:\images\...). Proměnná `fp` je hodnota výsledku funkce `fopen`. Funkce `fread` čte binární data souboru a uloží je do proměnné `Read_Dec`, parametr `uint8` převádí všechny znaky v souboru na osmibitová bezznaménková dekadická čísla. Vzniká tak vektor s dekadickými čísly, jehož délka odpovídá zároveň velikosti souboru. Funkce `fclose` pak zavře soubor. Proměnnou `Read_Dec` již není obtížné převést do hexadecimální formy.

Na obr. 3.2 je ukázka jednoduchého menu v aplikaci. Po kliknutí na položku *Otevřít soubor* se zobrazí okno (funkce uigetfile), které umožní výběr libovolného souboru typu GIF z pevného disku nebo z jiného média.

3.1.2 Ukládání bloků a jejich pozic

Pro úplný popis vnitřní struktury souboru typu GIF je nutné znát pozici a obsah jednotlivých bloků, ze kterých se celý soubor skládá. Informace o pozici je nutná k přehlednému vyznačení vybraného typu bloku v tabulce aplikace. Chceme-li znát detailní obsah bloku, je nutné celý blok uložit do vhodné proměnné. Na obr. 3.3 je algoritmus, pomoci kterého je hledaný typ bloku ve vektoru detekován. Celý princip ukládání jednotlivých bloků spočívá v tom, že se celý vektor `Read_Dec` s dekadickými hodnotami souboru cyklicky prochází prvek po prvku, přičemž je v něm hledána určitá posloupnost hodnot, která charakterizuje konkrétní hledaný typ bloku. Jakmile je blok identifikován, je uložena informace o jeho pozici i celý blok jako celek do příslušných proměnných. Pro každý typ bloku se vektor prochází zvlášť s výjimkou těch typů bloků, které mají předem známou fixní pozici v rámci souboru. Takové typy bloky jsou přímo uloženy do proměnných bez nutnosti

je jakýmkoli způsobem vyhledávat. Musíme však brát ohled na to, jak vyplývá z tabulky 2.2, že některé typy bloků se mohou v souboru vyskytovat víckrát. Takové typy bloků jsou ukládány do proměnné typu *cell*. Proměnná typu *cell* je v tomto případě jednorozměrné pole, jehož jednotlivými prvky jsou matice. Každá takováto matice představuje celý jeden hledaný typ bloku.



Obr. 3.3. Vývojový diagram pro ukládání bloků a jejich pozic

Následující text popisuje pořadí hledání jednotlivých typů bloků a způsob jejich detekce ve vektoru s dekadickými hodnotami. Jelikož je prohledáván soubor s dekadickými hodnotami, tak i podmínky v cyklech jsou koncipovány v dekadické formě. I když předchozí analýza interní struktury souboru GIF uvažovala všechny hodnoty bytů v hexadecimální formě, je efektivnější prohledávat vektor s dekadickými hodnotami. Je to z toho důvodu, že při převodu vektoru s dekadické do hexadecimální formy vzniká z vektoru o velikosti *Nx1* (s hodnotami *double* – dvojitá přesnost) stejně dlouhý vektor ovšem již o velikost *Nx2* (s hodnotami *char* – znaky). V případě, že chceme znát hodnotu např. patnáctého bytu ve vektoru v hexadecimálními hodnotami, tak při zápisu *Hex_Obsah(15)* je vrácen pouze první znak z dvojice znaků (např. *F* z dvojice *F9*). Proto je zpracování vektoru s hexadecimálními hodnotami nevhodné.

1. **Hlavička** - pozice a velikost hlavičky souboru jsou pevně dány. Informaci o pozici tedy není nutné ukládat. Do proměnně se uloží jen celý blok.

2. **Popis logické obrazovky** – pozice i velikost jsou fixní, stejný případ jako u hlavičky.

3. **Globální barvová paleta** – její přítomnost je testována v bloku popis logické obrazovky. Je-li přítomna, pak je její velikost vypočtena z prvních třech bitů pátého bytu bloku popis logické obrazovky. Její velikost je rovna dle vztahu 2.1. Jelikož se může v souboru vyskytovat maximálně jednou, je celý blok uložený do proměnné, která představuje pouhý vektor (matice o velikosti *m x 1*). Pozice bloku v rámci souboru má fixní hodnotu 14.

Zdrojový kód pro detekci a uložení výše uvedených typů bloků je v souboru *GIFstruc.m* ve funkci *doFile_Open_Callback*. Soubor se nachází na přiloženém CD. Princip ukládání bloků a jejich pozic do proměnných všech následujících bloků je stejný. Celý vektor `Read_Dec` se cyklicky prochází prvek po prvku, přičemž je hledána posloupnost hodnot charakterizující konkrétní hledaný typ bloku. Rozdíl spočívá pouze v hledaných posloupnostech. Všechny níže uvedené typy bloků se mohou vyskytovat v souboru víckrát, ukládají se tedy do proměnných typu *cell* a jejich pozice do proměnných představující vektor.

4. **Rozšíření grafické kontroly** – Zdrojový kód pro detekci a uložení tohoto typu bloku je v souboru *gce.m*, který se nachází na přiloženém CD.

5. **Aplikační rozšíření** – Zdrojový kód pro detekci a uložení tohoto typu bloku je v souboru *ae.m*, který se nachází na přiloženém CD.

6. **Rozšíření o jednoduchý text** – Zdrojový kód pro detekci a uložení tohoto typu bloku je v souboru *pte.m*, který se nachází na přiloženém CD.

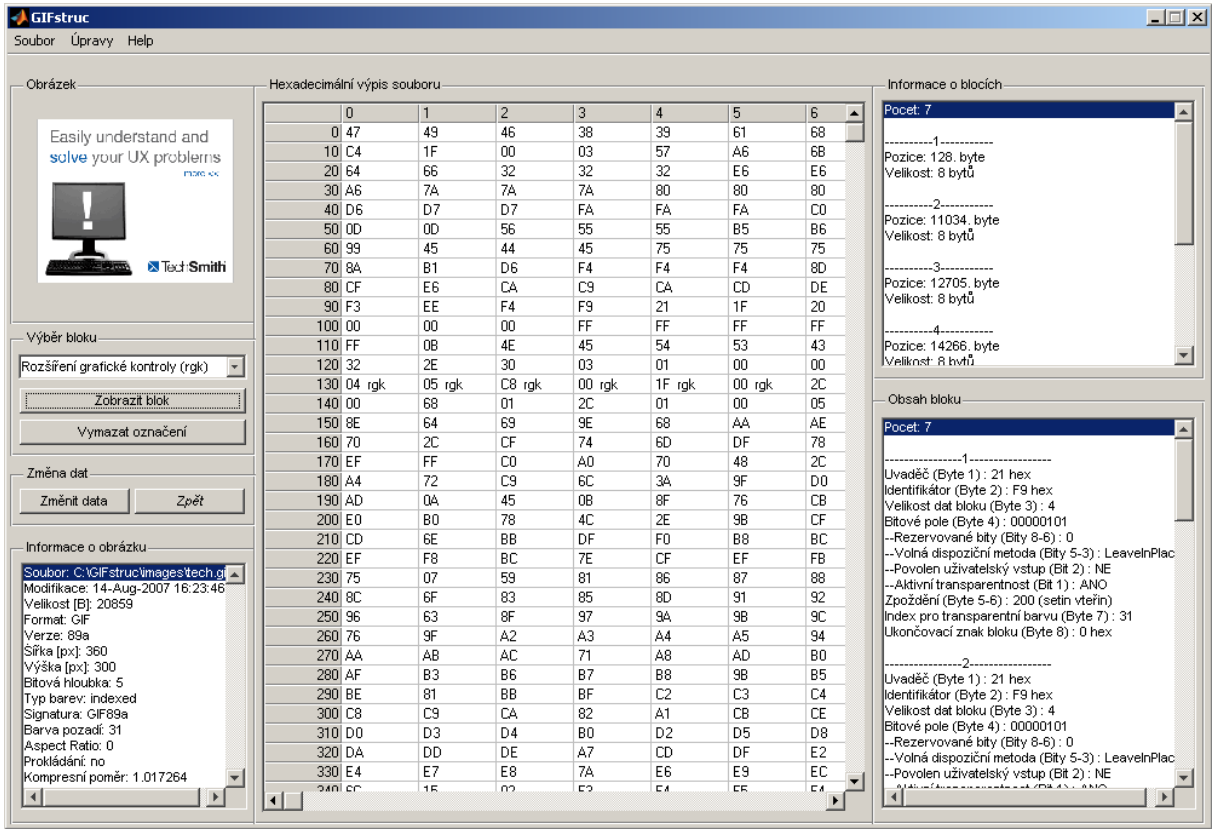
7. **Popis rámce, Lokální barvová paleta, Data rámce** – tyto bloky spolu úzce souvisí a tak jsou hledány současně. Vektor s dekadickými hodnotami se opět cyklicky prochází prvek po prvku, přičemž je hledán blok popis rámce. Dále se zjišťuje, zda je přítomna lokální barvová paleta. Je-li přítomna, je vypočtena její velikost a následně se zpracovávají samotná data rámce. Není-li lokální barvová paleta přítomna, rovnou se zpracovávají data rámce. Do proměnných jsou tedy uloženy jak jednotlivé bloky, tak jejich pozice v souboru. Zdrojový kód pro detekci a uložení těchto typů bloků je v souboru *GIFstruc.m* ve funkci *doFile_Open_Callback*. Soubor se nachází na přiloženém CD.

   Uložené bloky se samotnými daty rámce poslouží k výpočtu kompresního poměru. Všechny velikosti těchto bloků se sečtou a kompresní poměr použitého kompresního algoritmu se pak vypočte jako podíl velikosti souboru ku velikosti dat všech rámců. Informace o kompresním poměru je součástí informací o obrázku zobrazených v grafickém rozhraní,

8. **Rozšíření o komentář** – Zdrojový kód pro detekci a uložení tohoto typu bloku je v souboru *com.m*, který se nachází na přiloženém CD.

3.2 Návrh rozhraní

Velmi důležitou součástí při realizaci aplikace je navrhnutí vhodného rozhraní pro zobrazení získaných informací. Rozhraní musí být jednoduché, ale přitom účelné a názorné. Rozmístění jednotlivých komponent aplikace je uvedeno v příloze na obr. 7.1. Typy komponent byly voleny z ohledem na jejich funkci a vhodnost při zobrazování požadovaných informací. Celkový vzhled aplikace i se zobrazeným obrázkem je na obr. 3.4, v příloze na obr. 7.2 je pak tentýž obrázek v detailnější podobě a na obr 7.3, 7.4 a 7.5 jsou detailnější ukázky celé aplikace. Je vidět, že celé rozhraní aplikace lze rozdělit do několika částí. Postup výběru obrázku a zobrazení vybraného bloku lze shrnout do několika bodů. U každého bodu je jmenována i použitá komponenta:



Obr. 3.4. Celková aplikace

Soubor – velmi jednoduché menu umožňuje výběr libovolného obrázku a ukončení aplikace (viz obr. 3.2). Po kliknutí na položku *Otevřít soubor* se zobrazí okno, které umožní výběr libovolného souboru typu GIF z pevného disku nebo z jiného média. V okně je i nastaven filtr, takže kromě souborů s příponou *gif* není možné jiný typ souboru vybrat.

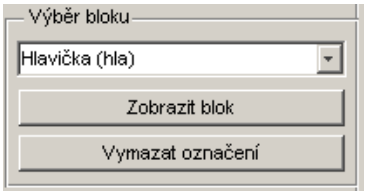
Obrázek – komponenta *axes* zobrazí vybraný obrázek. Program MATLAB neumožňuje zobrazit animovaný GIF jako sekvenci rámců s daným časovým zpožděním. Pouze umožňuje načíst všechny rámce animovaného obrázku za pomoci funkce *imread*. Z výsledku funkce ovšem není patrné, kolik rámců obrázek obsahuje. Animovaný obrázek je tedy zobrazen na základě znalosti počtu rámců vyplývajících z detekce bloků s daty rámců pomoci výše uvedených algoritmů. Celá animace (ovšem bez daného zpoždění uvedeném u každého rámce v bloku rozšíření grafické kontroly) je zobrazena pomoci cyklu *for* tak, že se jednotlivé rámce vykreslují do komponenty *axes,* první rámec je překryt druhým, druhý třetím atd.

Kromě komponenty *axes* se zvětšený obrázek zobrazí i v samostatné figuře. Zobrazení animovaného obrázku je principielně stejné jako u komponenty *axes*. Ze znalosti počtů rámců se pomoci cyklu *for* vykreslují jednotlivé rámce do jedné a té samé figury. Opět tak vzniká dojem animace.

Informace o obrázku – informace o základních vlastnostech vybraného obrázku lze získat jednoduchou funkcí *imfinfo*. Výstupem této funkce je proměnná typu *struktura*. Některé její položky spolu s kompresním poměrem jsou pak zobrazeny v komponentě *ListBox*.

Hexadecimální výpis souboru – tabulka (komponenta *VideoSoft FlexArray Control*) s hexadecimálním obsahem vybraného souboru. Každá buňka odpovídá jednomu bytu.

Výběr bloku – na obr. 3.5 je ukázka části rozhraní aplikace pro vyběr libovolného typu bloku a jeho vyznačení v tabulce. Za pomoci rozbalovacího menu (komponenta *PopUpMenu*) lze vybrat jakýkoliv typ bloku. Po kliknutí na tlačítko *Zobrazit blok* se příslušné byty odpovídající vybranému typu bloku v tabulce označí zkratkou v charakterizující vybraný typu bloku (viz obr. 7.2). Původním záměrem bylo hodnoty příslušných bytů barevně rozlišovat (pro každý typ bloku jiná barva). Bohužel tabulka neumožňuje měnit barvu písma textu v jednotlivých buňkách. Označovat byty zkratkami typů bloků je také výhodné v tom, že uživatel má přehled o hierarchii bloků v souboru. Např. blok rozšíření grafické kontroly předchází bloku popis rámce a ten předchází bloku data rámce. Kdyby se byty označovali pouze např. písmenem "x", uživatel by neměl dobrou představu o pořadí bloků. Tlačítko *Vymazat označení* veškeré označení bytů z tabulky odstraní.



Obr. 3.5. Část aplikace pro výběr typu bloku

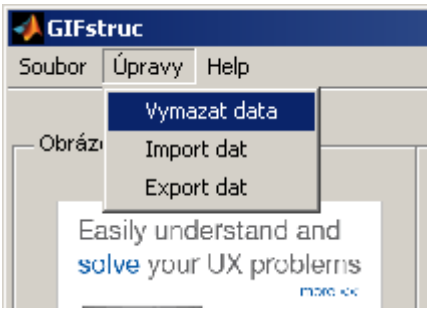
Informace o blocích – v této komponentě (*ListBox*) jsou zobrazovány informace o počtu a velikostech vybraného typu bloku.

Obsah bloku - v této komponentě (*ListBox*) je zobrazen detailní obsah vybraného typu bloku.

Některé v aplikaci použité komponenty vyžadují nadstandardní obsluhu co se týče zdrojového kódu. Např. komponenta *ListBox* není z hlediska plnění jejího obsahu příliš vhodná. K zobrazení dat je potřeba tyto data nejprve uložit do proměnné typu *cell* a pak je teprve zobrazit v této komponentě. Velké potíže působí i tabulka (*VideoSoft FlexArray Control*). Tabulka se nenachází v základních komponentách nabízených programem MATLAB, ale nachází se v skupině komponent s názvem *ActiveX*. Prakticky k ní neexistuje žádný návod jak s ní pracovat a proto je její obsluha velice obtížná. Zejména označování bytů v tabulce je velice náročné a velikost kódu je velká.

3.3 Funkce aplikace

Dosavadní funkce aplikace neposkytovali uživateli žádné možnosti jak s obrázkem jakýmkoli způsobem pracovat nebo ho dokonce měnit. Pro flexibilnější práci s obrázkem je tedy nutné zakomponovat do aplikace takové funkce, které mohou měnit data souboru vybraného obrázku. Na obr. 3.6 je jednoduché menu pro úpravu obrázku, pomoci kterého lze data ze souboru mazat nebo exportovat a importovat mezi jednotlivými obrázky. Další možností jak lze data změnit, je prostá změna hodnot bytů v tabulce s hexadecimálním obsahem souboru (viz. níže).



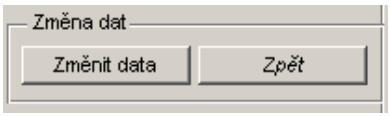
Obr. 3.6. Menu pro úpravu obrázku

Podstatou jakékoli změny dat souboru obrázku je obecně změna hodnot vektoru `Read_Dec`, který byl získán funkcí `fread` (viz. 3.1.1). Změnou tohoto vektoru vzniká vektor nový (`New_Vector`). Změnu obsahu souboru vysvětluje následující zdrojový kód:

```
fid = fopen(var, 'w');
fwrite(fid, New_Vector, 'uchar');
```

Stávající souboru se znovu otevře tentokrát s parametrem `'w'`, do souboru se bude zapisovat. Funkce `fwrite` celý soubor přepíše. Parametr `uchar` převádí všechna dekadická čísla nového vektoru na osmibitové bezznaménkové znaky. Takto modifikovaný soubor je znovu podroben algoritmům za odst. 3.1.2.

3.3.1 Změna dat

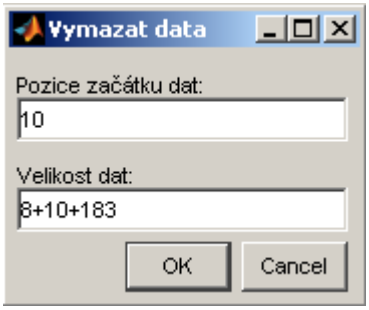


Obr. 3.7. Část aplikace pro změnu dat souboru

První možností, jak data v souboru měnit, je prostá změna hodnoty vybraného bytu. Nejjednodušším způsobem jak měnit data je prostřednictvím tabulky s hexadecimálním obsahem souboru. Na obr. 3.7 je ukázka části rozhraní aplikace pro změnu dat vybraného souboru. Tabulka umožňuje editaci jednotlivých buněk a proto stačí pouhým přepsáním hodnoty odpovídajícího bytu změnit obsah celého souboru. Po kliknutí na tlačítko *Změnit data* se celý obsah souboru přepíše aktuálním obsahem tabulky a vznikne tak nový pozměněný soubor. Funkce tlačítka *Zpět* bude vysvětlena později.

Tímto způsobem je možné měnit barvy obrázku pouhou změnou hodnot v aktuální barvové paletě (globální příp. lokální). Je nutné si však uvědomit (viz. 2.7.5 a 2.7.7), že aktivní barvová paleta sestává s tripletů RGB a proto ke změně jedné barvy obrázku je třeba změnit hodnoty tří bytů. Je možné měnit i barvu pozadí obrázku změnou indexu v bloku popisující logickou obrazovku. U animovaných obrázků je pak možné měnit rychlost překreslování jednotlivých rámců a lze tak libovolně určovat rychlost celé animace.

3.3.2 Vymazání dat



Obr. 3.8. Dialogové okno pro export dat

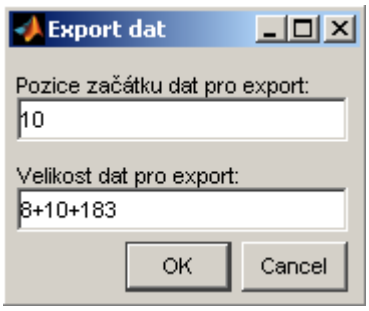
Další možností, jak interní data souboru pozměnit, je mazání dat. Funkce není omezena na vymazání pouze jediného bloku, ale poskytuje uživateli možnost vymazat jakkoli velký souvislý úsek dat. V dialogovém okně na obr. 3.8 požaduje aplikace po uživateli pouze zadání začátku úseku (pozici prvního bytu úseku) a velikost úseku dat, který má být ze souboru vymazán. Velikost úseku dat může být zadána formou výrazu. Je to velice výhodné, protože v případě, že bychom chtěli například smazat více rámců, museli bychom jejich velikosti sčítat a výsledný součet zadat do pole v dialogovém okně.

Uživatel tedy může vymazat ze souboru komentáře, které mnohdy zbytečně zvětšují jeho velikost. Z animované obrázku je možné smazáním všech rámců vyjma jednoho vytvořit neanimovaný GIF, přičemž výběr rámce, který v obrázku zůstane je volbou uživatele.

3.3.3 Export/Import dat

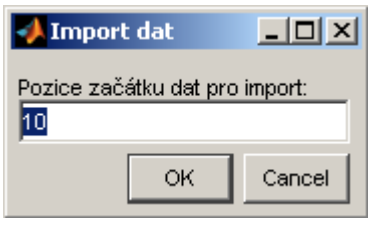
Velice atraktivní funkcí aplikace je možnost exportu a importu dat mezi obrázky. Z načteného obrázku lze pomoci dialogového okna na obr 3.9 vybrat souvislou část dat zadáním pozice začátku (pozice prvního bytu vybraných dat) a velikostí dat pro export.

Vybraná data jsou uložena do souboru *data.mat*. Při otevření a načtení jiného obrázku lze data do tohoto obrázku importovat pomoci dialogového okna na obr 3.10. Uživatel pouze zadá pozici, na kterou mají být data vložena do souboru. Po kliknutí na tlačítko *OK* se data určená pro import ze souboru *data.mat* vloží na zadanou pozici. Tímto způsobem lze vytvářet jak obrázky s více rámci, tak i animované.



Obr. 3.9. Dialogové okno pro export dat

Vkládání rámců jednoho obrázku do druhého obrázku má ovšem řadu omezení. V prvé řadě se může stát, že bude-li importovaný rámec rozměrově větší než obrázek, do kterého se má vložit, tak MATLAB není schopen takto vzniklý obrázek zpracovat. Dalším omezení je při vytváření animací nebo obrázků s více rámci z již existujících obrázků, které neobsahují stejné barvy. Musíme brát v potaz tu skutečnost, že ne všechny obrázky mají stejnou barvovou paletu. Jak již bylo řečeno, data rámce obsahují zakódované indexy do aktivní barvové palety. V praxi to znamená, že vložíme-li rámec jednoho obrázku do obrázku, který nemá úplně stejnou barvovou paletu jako zdrojový obrázek, bude vložený rámec vykreslen jinými barvami. Je to zcela logické, protože zakódované indexy vloženého rámce odkazují na barvy, které jsou jiné než ve zdrojovém obrázku. Na obr. 3.11 je ukázka změny barvy vloženého obrázky do obrázku jiného. Jak je vidět, je lepší si vytvořit vlastní obrázky typu GIF (např. pomoci obyčejného malování) se stejnými barvami a pomoci aplikace pak operovat s rámci bez změny barev. Další omezení vyplývá z určité nutnosti dodržet hierarchii rámců v souboru. Pokud exportujeme do souboru rozměrově menší rámec před rámec větší, tak tento vložený rámec ve výsledku nebude vidět, protože bude větším rámce zastíněn. Poslední omezení opět úzce souvisí s barvovou paletou. Pokud budeme importovat rámec z obrázku, který má větší barvovou paletu (v podstatě má obrázek větší bitovou hloubku) do obrázku s menší barvovou paletou, tedy s menším počtem barev, MATLAB opět není schopen takto vzniklý obrázek zpracovat. Zakódované indexy vloženého rámce by v tomto případě odkazovali na pozice v barvové paletě, které neexistují.



Obr. 3.10. Dialogové okno pro import dat

Obr. 3.11. Obrázek zcela vlevo je vložen do prostředního obrázku. Výsledek exportu obrázku je pak zobrazen zcela vpravo

Výše uvedené funkce umožňují nezkušenému uživateli takové úpravy dat souboru, které mohou mít špatný dopad na funkčnost celého obrázku. Např. uživatel může vymazat blok, který se musí nacházet v každém validním souboru (viz. tabulka 2.2) nebo se může pokusit změnit hodnoty bytů, které musí zůstat neměnné. V takovém případě aplikace pouze ohlásí chybové hlášení bez přesnějšího udání, k jaké chybě došlo. Jelikož se jedná o výukovou aplikaci, je uživatel sám zodpovědný za to, jaká data mění. Aby nedošlo k nevratnému znehodnocení obrázku, je možné pomoci tlačítka *Zpět* vrátit obsah souboru do původní podoby.

V případě že si uživatel není jistý, která data může měnit, může využít nápovědy. Po kliknutí na tlačítko *Help* se otevře textový soubor s podrobným popisem interní struktury grafického formátu GIF.

3.4 Úlohy ke cvičení

Aplikace se otevře v programu MATLAB spuštěním souboru *GIFstruc.m* v adresáři *GIFstruc*. Obrázky určené pro úlohy a výsledné kontrolní obrázky se nachází v adresáři *Zadání*.

1. Zobrazte obrázek *vlajka.gif* a prohlédněte si obsah souboru

2. Změňte barvu obrázku tak, aby se z bílé barvy na vlajce stala barva zelená
   - Pomoci rozbalovacího menu vybereme blok *globální barvová paleta*. Po kliknutí na tlačítko *Zobrazit blok* se příslušné byty v tabulce označí zkratkou a zároveň se barvová paleta zobrazí do zvláštní figury. Podle této palety poznáme, kterou barvu (triplet) je třeba změnit. V tomto případě je nutné změnit první triplet. Tomuto tripletu odpovídají v tabulce byty na pozicích 13, 14 a 15. Pro zelenou barvu je tedy nutné změnit hodnoty bytů na hodnoty 0, FF, 0 (hodnoty musí být v hexadecimálním tvaru). Po kliknutí na tlačítko *Změnit data* se soubor přepíše a změna barvy bude patrná. Pokud změna barvy proběhla v pořádku, měl by být obrázek *vlajka.gif* stejný jako obrázek *result2.gif*.

3. Vymažte ze souboru komentář
   - Zobrazíme blok *rozšíření o komentář*. Informace o pozici a velikosti je zobrazena v části aplikace s názvem *Informace o blocích*. V menu *Úpravy* vybereme položku *Vymazat dat*. Do zobrazeného dialogového okna zadáme pozici začátku dat (248) a velikost dat (243). Po kliknutí na tlačítko *OK* se komentář ze souboru vymaže. Pokud odstranění komentáře proběhlo v pořádku, měl by obsah souboru *vlajka.gif* odpovídat souboru *result3.gif*.

4. Exportujte celý rámec ze souboru *vlajka.gif* a importujte ho do souboru *logo_fekt.gif*
   - Necháme načtený obrázek *vlajka.gif*. V menu *Úpravy* vybereme položku *Export dat*. Zobrazené dialogové okno požaduje zadání pozice začátku dat a velikost dat pro export. Chceme-li exportovat celý rámec, musíme exportovat jak blok *popis rámce*, tak i blok *data rámce*. Pro zjištění pozice začátku celého rámce, tedy pozice začátku dat pro export, stačí znát pozici bloku *popis rámce*. Zobrazíme tedy blok *popis rámce*, jeho pozice v souboru je 61. Velikost tohoto bloku je 10 bytů. K této hodnotě ještě musíme přičíst velikost bloku *data rámce*. Pokud tento blok zobrazíme, tak zjistíme, že je jeho velikost 177 bytů. Velikost dat pro export je tedy 10+177 bytů a tento výraz zapíšeme do odpovídajícího pole v dialogovém okně. Po kliknutí na tlačítko *OK* se rámec uloží do souboru *data.mat*. Otevřeme obrázek *logo_fekt.gif*. V menu *Úpravy* vybereme položku *Import dat*. V dialogovém okně zadáme pozici, na kterou se má rámec vložit. Nejvhodnější pozice pro import rámce je hned za blok *data rámce* obrázku *logo_fekt.gif*. Zobrazíme tento blok. Do dialogového okna tedy zadáme hodnotu 3932. Po kliknutí na tlačítko *OK* se rámec do souboru importuje. Pokud export rámce proběhl v pořádku, měl by být obrázek *logo_fekt.gif* stejný jako obrázek *result4.gif*. Diskutujte, proč je vložený obrázek vykreslen jinými barvami.

V tomto bodě je nutné podotknout, že vzniklý obrázek nejsou některé prohlížeče schopné zobrazit. Zejména pak prohlížeč obrázků a faxů v OS Windows. Je tedy třeba použít jiný prohlížeč např. i obyčejný Explorer. Platí to i pro bod 5.

5. Změňte pozici právě vloženého rámce tak, aby byl zhruba uprostřed obrázku
   - Informace o pozici rámce uvnitř logické obrazovky je dána v bloku *popis rámce*. Pozice rámce v logické obrazovce se vztahuje k jeho levému hornímu rohu. Zobrazíme tedy blok *popis rámce*. Obrázek v tomto případě obsahuje dva rámce. U druhého vyznačeného bloku změníme hodnotu 2,, 3,, 4. a 5. bytu. V tabulce jim odpovídají byty na pozicích 3933, 3934, 3935 a 3936. Byty na pozicích 3933 a 3934 (3933. byte – *"little endian"*, 3934. byte – *"big endian"*) určují pozici levého okraje rámce, byty na pozicích 3935 a 3936 (3935. byte – *"little endian"*, 3936. byte – *"big endian"*) určují pozici horního okraje rámce. Stačí tedy tyto čtyři byty nastavit na hodnoty 28, 0, 30, 0 (hodnoty musí být v hexadecimálním tvaru) aby byl rámec zhruba uprostřed obrázku. Klikneme na tlačítko *Změnit data*. Pokud změna pozice rámce proběhla v pořádku, měl by být obrázek *logo_fekt.gif* stejný jako obrázek *result5.gif*.

6. Načtěte a zobrazte obrázek *oci.gif* a prohlédněte si obsah souboru

7. Zmenšete časové zpoždění všech rámců
   - Údaj o časovém zpoždění každého rámce je uveden v bloku *rozšíření grafické kontroly* a to v 5. a 6. bytu (5. byte – *"little endian"*, 6. byte – *"big endian"*) tohoto bloku. Zobrazíme tedy tyto bloky (animovaný obrázek má více rámců a proto bude počet těchto bloků odpovídat počtu rámců). Změnou hodnot (hodnoty musí být v hexadecimálním tvaru) příslušných bytů tedy můžeme zmenšit zpoždění rámců. Kliknutím na tlačítko *Změnit data* se celý souboru opět přepíše novými hodnotami. Jak již bylo řečeno, aplikace při zobrazování animovaných obrázků nebere ohled na časové zpoždění rámců. Je tedy vhodné si prohlédnout upravený animovaný obrázek pomoci nějakého standardního prohlížeče.

8. Vytvořte z animovaného obrázku *oci.gif* obrázek neanimovaný
   - Smazáním všech rámců vyjma jednoho lze vytvořit neanimovaný obrázek obsahující pouze jeden rámec. Pro jednoduchost ponecháme v obrázku *oci.gif* první rámec a všechny ostatní smažeme. Kromě samotných bloků *popis rámce* a *data rámce* je nutné smazat i všechny bloky *rozšíření grafické kontroly*, které oběma typům bloků předcházejí. Zobrazíme si tedy bloky *rozšíření grafické kontroly*. Pozice začátku zobrazeného druhého bloku je tedy rozhodující a použijeme ji jako hodnotu určující pozici (pozice 826) začátku dat v dialogovém okně pro vymazání dat. Zadání velikosti dat pro vymazání je nyní poněkud obtížnější. Musíme si uvědomit, že v tomto případě mažeme tři rámce, tj. tři bloky *popis rámec*, každý o velikosti 10B a tři bloky *rozšíření grafické kontroly*, každý o velikosti 8B. K tomu je třeba připočítat velikosti tří bloků *data rámce*. Tyto velikosti získáme jejich zobrazením. Konečná velikost dat pro vymazání tedy odpovídá: 3*10+3*8+239+264+169, tento výraz zapíšeme do příslušného pole v dialogovém okně. Po kliknutí na tlačítko *OK* se vybraná data ze souboru vymažou. Pokud odstranění rámců proběhlo v pořádku, měl by být obrázek *oci.gif* stejný jako obrázek *result8.gif*.

9. Vytvořte animovaný GIF. Využijte k tomu obrázků *b.gif*, *m.gif*, *s.gif* a *d.gif*
   - Podstata vytvoření animovaného obrázku spočívá v importu rámců z obrázků *m.gif*, *s.gif*, a *d.gif* do obrázku *b.gif* (ve finále bude tedy obrázek *b.gif* animovaný). U

každého rámce je nutné nastavit zpoždění v bloku *rozšíření grafické kontroly*. Jelikož u těchto obrázků není tento blok přítomen, je prvotním úkolem nejdříve tento blok do jednotlivých souborů obrázků vložit. Využijeme obrázku *oci.gif*, ze kterého exportujeme blok *rozšíření grafické kontroly* a importujeme ho do všech souborů (*b.gif*, *m.gif*, *s.gif* a *d.gif*) tak, aby byl umístěn hned před blok *popis rámce* (blok rozšíření grafické kontroly musí odpovídat přesně začátku bloku *popis rámce*, konkrétně v obrázku *b.gif* vložíme tento blok na pozici 781*)*. Nyní budeme postupně vkládat rámce z obrázků *m.gif*, *s.gif* a *d.gif* do obrázku *b.gif* tak, aby vložené rámce byly v souboru *b.gif* umístěny hned za sebou. Exportujeme rámec (tedy bloky *rozšíření grafické kontroly* + *popis rámce* + *data rámce*) z obrázku *m.gif* a importujeme jej hned za blok *data rámce* obrázku *b.gif* na pozici 979. Za nově vložený rámec importujeme stejným způsobem rámec z obrázku *s.gif* a za ten hned rámec z obrázku *d.gif*. Pokud import všech rámců proběhl v pořádku, měl by být obrázek *b.gif* stejný jako obrázek *result9.gif*. Opět je nutné výsledný obrázek zobrazit pomoci standardního prohlížeče, který respektuje zpoždění rámců.

Výsledný obrázek se oproti obrázku *result9.gif* může lišit rychlostí animace. V bodě 7. jsme měnili libovolně rychlost animace obrázku *oci.gif* a z tohoto obrázku jsem potom exportovali blok *rozšíření grafické kontroly* do všech obrázků použitých v animaci v bodě 9. Je proto jasné, že zpoždění dané v bloku *rozšíření grafické kontroly* se přenese i do výsledné animace,

4 Závěr

Hlavním cílem celé práce bylo navrhnout a realizovat výukovou aplikaci pro přehledné zobrazení interní struktury souboru typu GIF. Jako vývojové prostředí byl zvolen program MATLAB GUI. Nezbytně nutným předpokladem pro správnou a úspěšnou realizaci aplikace byla dokonalá znalost interní struktury souboru typu GIF. Bylo zapotřebí velmi důkladně prostudovat informační zdroje uvedené v bibliografii. Bylo důležité znát a pochopit nejen funkci a podrobný obsah všech typů bloků, ale zejména znát i jejich hierarchii. Vzhledem k tomu, že soubor GIF obsahuje poměrně větší množství typů bloků, které se mohou v souboru i několikrát opakovat a mohou být umístěny na různých pozicích v souboru, bylo zapotřebí zvážit všechny možnosti ke správnému návrhu algoritmů pro detekci jednotlivých typů bloků.

Celé grafické rozhraní aplikace bylo navrženo tak, aby bylo pro uživatele co možná nejjednodušší a nejpřehlednější. Při jeho návrhu hrála rozhodující roli znalost funkce a obsluhy všech použitých komponent. Typy a umístění komponent v grafickém rozhraní byly voleny tak, aby co nejvíce vyhovovali požadované funkci aplikace. Jak již bylo řečeno, obsluha některých typů komponent byla velmi obtížná a zdlouhavá, zejména pak obsluha komponenty VideoSoft Flexarray Control. Jako výchozí informační zdroj posloužil Help v programu MATLAB.

Realizovaná aplikace velmi přehledně zobrazuje základní parametry vybraného obrázku a velmi názorně vysvětluje, kde a jaké typy bloků se v tomto souboru vyskytují. Uživatel má tak dokonalý a kompletní přehled o stavbě interní struktury souboru vybraného obrázku. Navíc je možné zobrazit i globální či lokální paletu barev ve zvláštní figuře a mít tak představu o všech barvách, které obrázek obsahuje. Aplikace je konstruována tak, aby se uživatel pomoci jednoduchých operací seznámil s interní strukturou souboru. Operace s obrázky, které aplikace umožňuje, nelze úspěšně provádět bez znalosti obsahu, funkce a pozic jednotlivých typů bloků. Proto je celá aplikace koncipována jako výukový program v oblasti počítačové grafiky. Po úspěšném zvládnutí úloh z odst. 3.4 si uživatel prohloubí znalosti o interní struktuře souboru typu GIF, což je splnění cíle této práce.

Bibliografie

[1] ČANDLÍK, Marek. *Fraktálové kódovanie obrazov*, OFTIS Ostrava, 2005. 100 s. ISBN 80-900745-3-7

[2] JAN, Jiří. *Číslicová filtrace, analýza a restaurace signálů*, Nakladatelství Vutium, Vysoké učení technické v Brně, 2002. 427 s. ISBN 80-214-1558-4

[3] DOYLE, Matt. *Understanding image formats*. Elated [online]. Leden 1998 [cit. 19. března 2007] Dostupné na WWW: < http://www.elated.com/articles/understanding-image-formats >.

[4] SOJKA, Eduard. Digitální zpracování a analýza obrazů, Ostrava VŠB – Technická univerzita, 2000. 133s. ISBN 80-7078-746-5

[5] CompuServe Incorporated, *Graphics Interchange Format(sm)* [online]. 31 July 1990 [cit. 10. října 2007]. Dostupné na WWW: <http://www.w3.org/Graphics/GIF/spec-gif89a.txt>

[6] TIŠŇOVSKÝ, Pavel. Případ GIF. *Root.cz, informace nejen ze světa Linuxu* [online]. 2006, [cit. 2007-11-10]. Dostupný na WWW:< http://www.root.cz/clanky/jpeg-kral-rastrovych-grafickych-formatu/>. ISSN 1212-8309

[7] LYNCH, Patrick., HORTON, Sarah. *Web style guide, 2nd edition* [online]. 2005, [cit. 2007-15-10]. Dostupný na WWW:< http://webstyleguide.com/graphics/gifs.html>.

6 Seznam zkratek

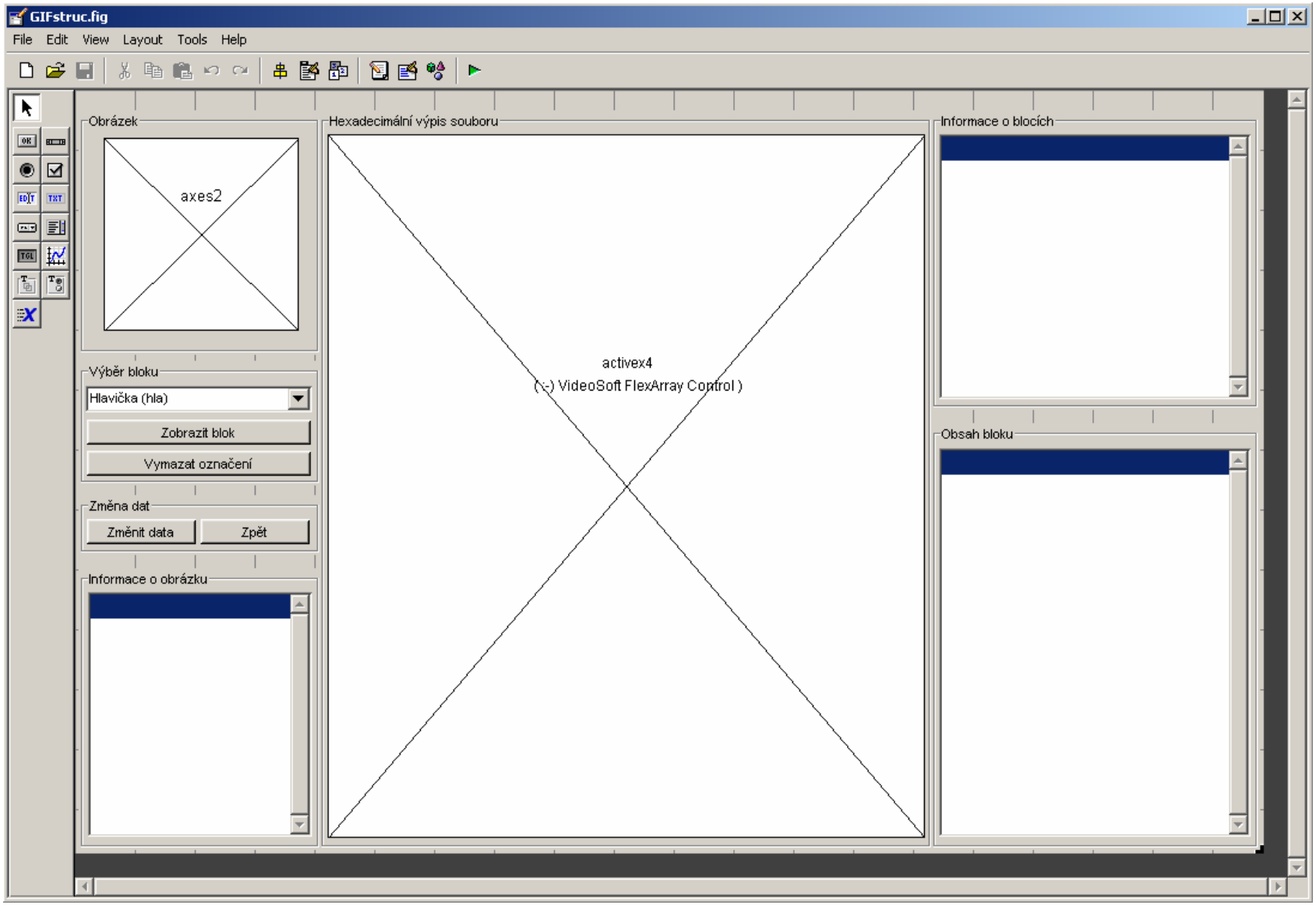
GIF	Graphics Interchange Format
BMP	Microsoft Windows Bitmap
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
TIFF	Tagged Image File Format
LZW	Lempel-Ziv-Welch
DPCM	Differential Pulse Code Modulation
DCT	Discrete Cosine Transform
ZIP	Souborový formát pro kompresi a archivaci dat
ARJ	Archived by Robert Jung
LZ77	Lempel-Ziv 77
RLC	Run Length Coding; někdy též RLE (Run Length Encoding)
ASCII	American Standard Code for Information Interchange
AC	Alternating Current
DC	Direct Current
RGB	Red-Green-Blue
YCbCr	Y- luma component, Cb - blue component, Cr - red chroma component
YUV	Y-luma component; U, V chrominance
YIQ	Y-luma component, I-in-phase, Q-quadrature
MATLAB	Matrix Laboratory
GUI	Graphical User Interface
MSB	Most significant bit
LSB	Least significant bit
2D	Two dimension

7 Přílohy

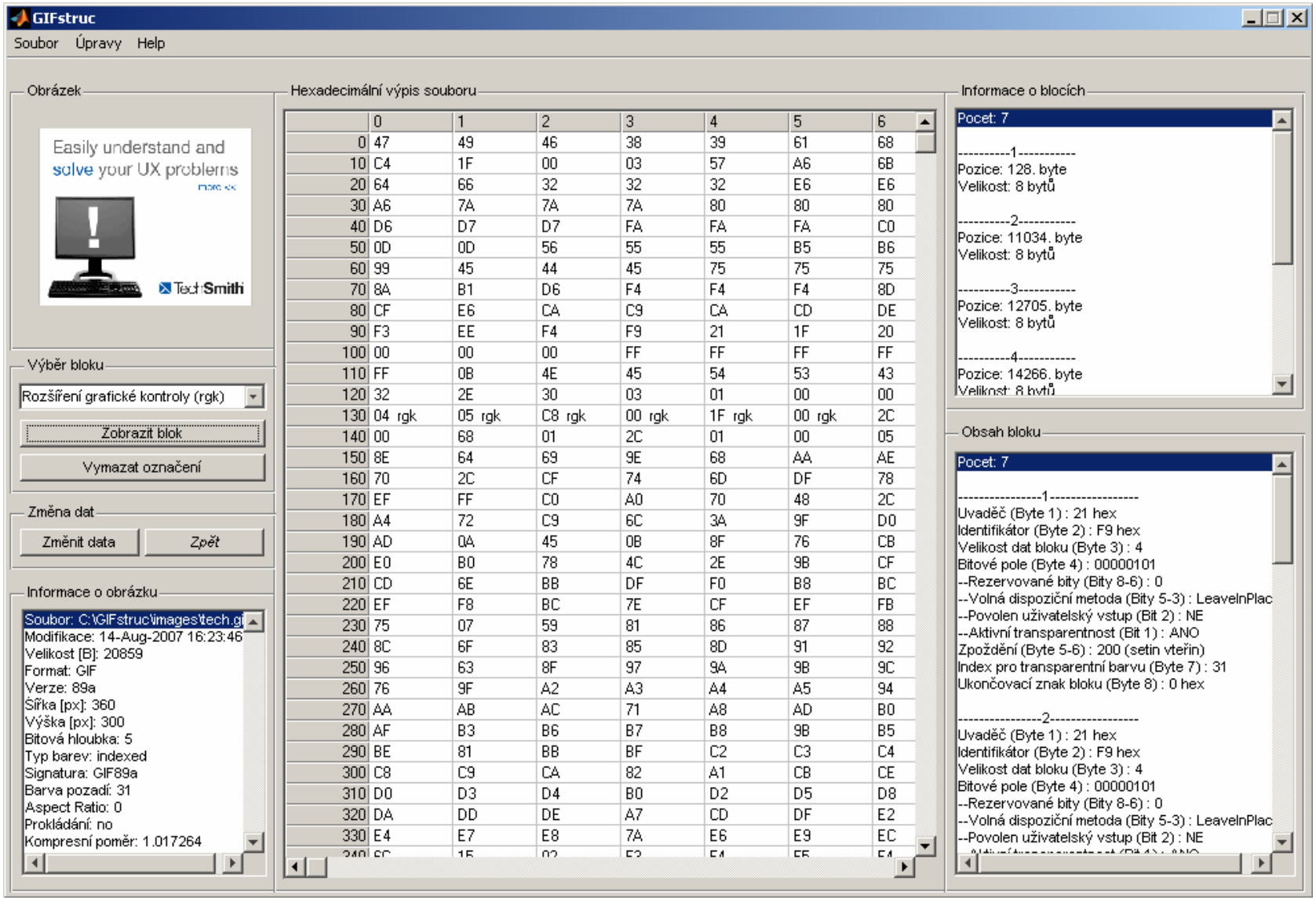
Zdrojový kód celé aplikace se nachází na přiloženém CD v adresáři *GIFstruc*. V tomto adresáři se nachází zdrojový kód pro detekci všech výše uvedeným typů bloků:

Rozšíření grafické kontroly – soubor *gce.m*
Rozšíření o komentář – soubor *com.m*
Rozšíření o jednoduchý text – soubor *pte.m*
Aplikační rozšíření – *ae.m*
Popis rámce, data rámce, hlavička, lokální a globální barvová paleta, popis logické obrazovky a ukončovací znak souboru – souboru *GIFstruc.m* ve funkci *doFile_Open_Callback*.

V adresáři *GIFstruc* se nachází i složka *images*, která obsahuje obrázky k testování aplikace a ve složce *Zadání* jsou pak potřebné obrázky k úlohám z odst. 3.4.

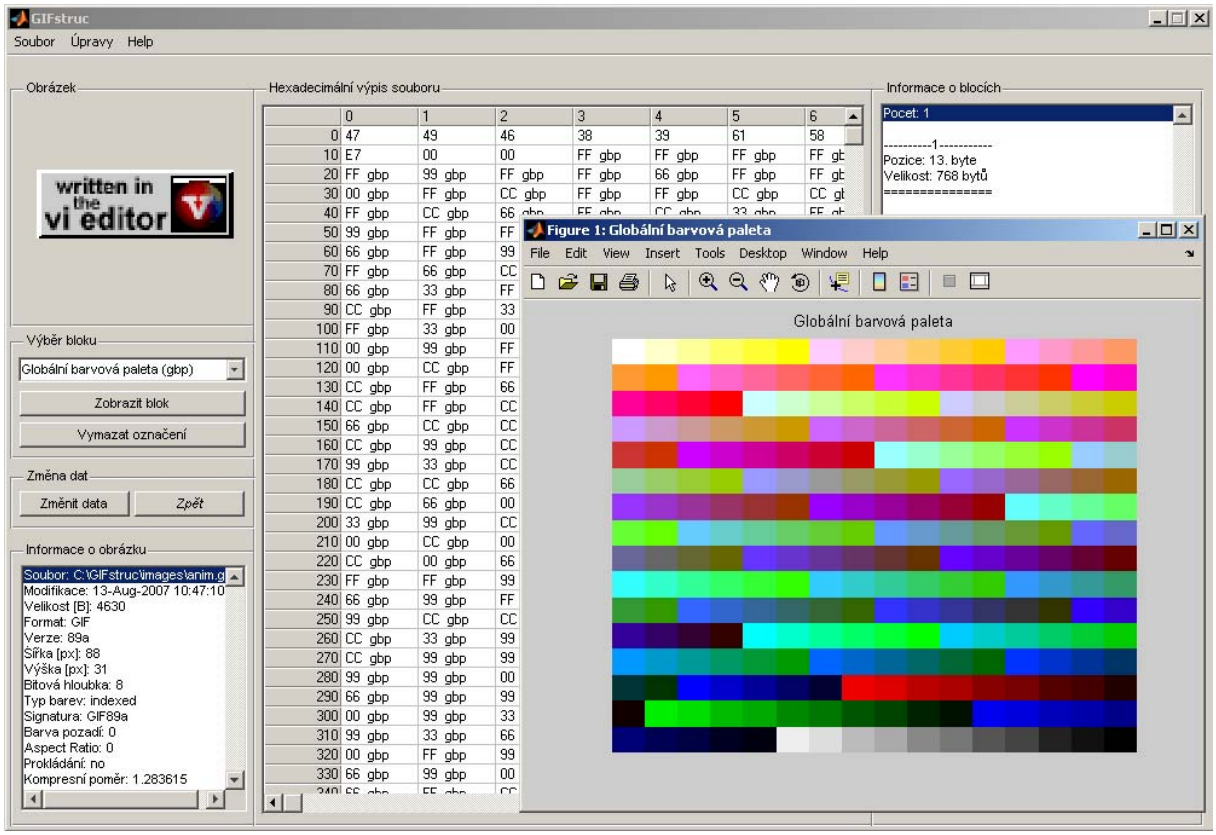


Obr. 7.1. Rozmístění a typy komponent aplikace

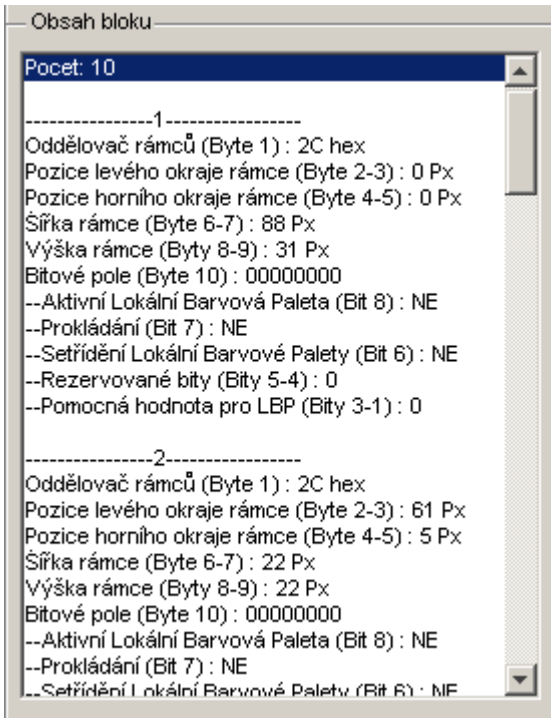


Obr. 7.2. Celková aplikace

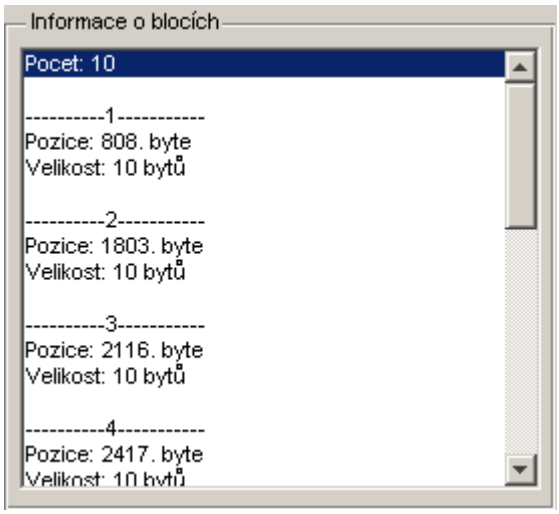
Detailnější ukázky aplikace



Obr. 7.3. Ukázka zobrazené globální barvové palety



Obr. 7.4. Obsah bloku *popis rámce* náležící obrázku z obr. 7.3



Obr. 7.5. Počet, pozice a velikost bloku *popis rámce* z obr. 7.3