Univerzita Pardubice

Fakulta elektrotechniky a informatiky

Vybrané algoritmy pro práci s rastrovými obrazy

Jiří Novák

Bakalářská práce
2008

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra informačních technologií
Akademický rok: **2007/2008**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Jiří NOVÁK**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Vybrané algoritmy pro práci s rastrovými obrazy.**

Zásady pro vypracování:

Cíl: implementace vybraných rastrových algoritmů a jejich použití v prgramu, umožňující specifické skládání obrazů.
Teoretická část:
* Popis algoritmů pro skládání částí obrazu.
* Složitost a výpočetní náročnost použitých algoritmů.
* Porovnání alternativních postupů.
* Popis algoritmů pro sjednocení DPI, převzorkování, tvorbu adaptivní palety, formát vybraných grafických formátů.

Implemetační část:
* Vlastní implementace vybraných algoritmů.
* Vytvoření programu pro specifické skládání obrazu a jeho editaci za použití výše uvedených algoritmů.
* Porovnání s hotovými řešeními v Delphi.

Požadavky:
* Jednoduchost, User-Friendly.
* Implementace v Object Pascal/Delphi.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

ŽÁRA, Jiří, BENEŠ, Bedřich, SOCHOR, Jiří, FELKEL, Petr. Moderní počítačová grafika. 2. přepracované vydání. Brno : Computer Press, 2004. ISBN 80-251-0454-0.

FOLEY, D. James, van DAM, Andries, FEINER, Steven, HUGHES, John. Computer Graphics – Principles and Practice. New York : Addison-Wesley, 1990. ISBN 0-201-84840-6.

Vedoucí bakalářské práce: **Ing. Petr Veselý**
Katedra informatiky v dopravě

Datum zadání bakalářské práce: **30. listopadu 2007**

Termín odevzdání bakalářské práce: **16. května 2008**

Univerzita Pardubice Česká republika

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

V Pardubicích dne 29. dubna 2008

Abstrakt

Pro využití v praxi je třeba vytvořit aplikaci se specifickými vlastnostmi, které vývojové prostředí standardně neposkytuje. Jedná se o implementaci několika algoritmů z oblasti počítačové grafiky a o rozšíření možností práce s různými grafickými formáty. V problematice algoritmů se práce konkrétně zaměřuje na vytvoření adaptivní palety (kapitola 2), specifické spojení dvou rastrových obrazů (kapitola 3) a vybrané techniky interpolace (kapitola 4). Rozšíření o možnost práce s nejběžnějšími grafickými formáty a hodnotou DPI obrazu je věnována kapitola 5.

Klíčová slova

počítačová grafika, adaptivní paleta, interpolace, grafické formáty, rastrová grafika

Abstract

It is necessary to make, for a practically usage, an application with specific properties which are not provided in the research environment as a standard. It is an implementation of several algorithms in the computer graphics sphere and extension of opportunities while working with various graphic formats.

In the algorithm issue, the paper is focused specifically on creating an adaptive palette (Chapter 2), specific connection of two raster pictures (Chapter 3) and a selected interpolation technique (Chapter 4). Chapter 5 deals with extension with an opportunity to work with the most common graphic formats and DPI value.

Keywords

computer graphic, adaptive palette, interpolation, graphic formats, raster graphic

Poděkování

Tímto chci poděkovat všem lidem, kteří přispěli k vypracování této práce, především pak vedoucímu práce, panu Ing. Petru Veselému za pomoc při řešení dílčích úkolů, za čas, po který se mi věnoval v rámci odborných konzultací a za poskytnutí studijních materiálů.

Obsah

Seznam obrázků

1 Úvod

Cílem této práce je vytvořit aplikaci na specifické spojení obrazů, která bude „ušitá na míru“ pro praktické využití[1]. Jako vývojové prostředí pro tuto aplikaci bude použito Borland Delphi 7 Personal Edition. Vedle stěžejního úkolu naprogramovat toto spojení, je také důležitá celková představa aplikace, zahrnující několik dílčích problémů, které je třeba vyřešit. Pro většinu těchto problémů prostředí Borland Delphi nenabízí standardní řešení. Další součástí práce je pokusit se implementovat vybrané algoritmy počítačové grafiky. Některé úlohy mohou mít více řešení, tudíž je třeba jednotlivé způsoby implementace porovnat z hlediska časové a výpočetní náročnosti a na základě těchto údajů zvážit, která metoda bude zvolena jako nejvhodnější. Ta potom bude zapojena do hlavní aplikace.

1.1 Forma obsahu kapitol

Každá kapitola obsahuje několik dílčích částí; první se na téma kapitoly zaměřuje z teoretického hlediska a je nazvána *Teoretický rozbor*. Následuje část *Návrh řešení*, kde je navrženo jedno, nebo více způsobů, jak konkrétní úkol realizovat. Konkrétní realizace je potom popsána v části Implementace. Následuje shrnutí kapitoly v části D*osažené výsledky*.

1.2 Hlavní cíle práce

1.2.1 Adaptivní paleta

Tato kapitola je zaměřena na převedení obrazu o barevné hloubce 24 bitů na obraz, využívající paletu o 256 barvách (8 bitová grafika). Cílem je implementovat algoritmus na vypočítání této palety a její aplikace na obraz.

1.2.2 Spojení obrazů

Jedná se o hlavní úkol aplikace; cílem je spojit dva obrazy do výsledného po proužcích tak, aby se velikost proužku dala nastavovat. Bude třeba nalézt takové řešení, které bude dostatečně rychlé a efektivní.

1 tornado cz, a. s., Hůrka 1798, 530 12 Pardubice www.tornado.cz

1.2.3 Interpolace

Pro měnu velikosti obrazů se využívá celá řada interpolací. Cílem je naprogramovat vybrané metody, porovnat a vybrat tu, která bude pro potřeby aplikace nejvhodnější.

1.2.4 Grafické formáty, DPI

Borland Delphi standardně pracují pouze s formátem BMP, což je pro potřeby aplikace. Dále toto prostředí nenabízí možnost práce s hodnotou DPI obrazu. Cílem je nalézt ze všech dostupných zdrojů co nejvíce modulů a algoritmů, které aplikaci rozšíří o další formáty a práci s jejich hodnotou DPI.

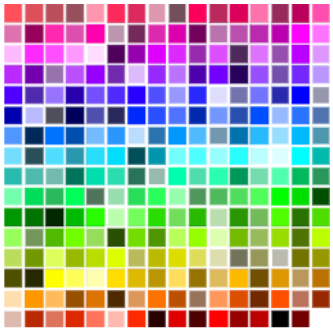
2 Adaptivní paleta

2.1 Teoretický rozbor

Obrazy, které mají grafickou hloubku 8 bitů mohou obsahovat maximálně 256 barev. Otázkou zůstává, jak takto omezený barevný prostor co nejefektivněji využít. Jako obecný výběr z celého RGB prostoru je často uváděna standardní 3-3-2 paleta (Obr. 1). Způsob tohoto rozložení je znázorněno v následujícím výpočtu (Vzorec 1). Pro názorné ukázky je použit příkladový obrázek (Obr. 2)[2].

$$R: 3 \Rightarrow 2^3 = 8 \text{ úrovní}$$
$$G: 3 \Rightarrow 2^3 = 8 \text{ úrovní}$$
$$B: 2 \Rightarrow 2^2 = 4 \text{ úrovně}$$
$$\text{Kombinace}: R \cdot G \cdot B = 8 \cdot 8 \cdot 4 = 256$$

Vzorec 1



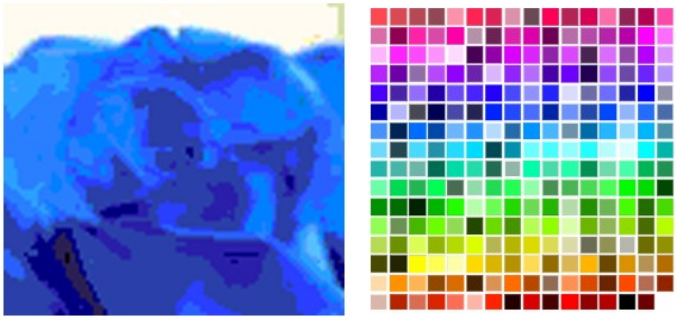
Obr. 1: Standardní 3-3-2 paleta

Obr. 2: Originální obraz

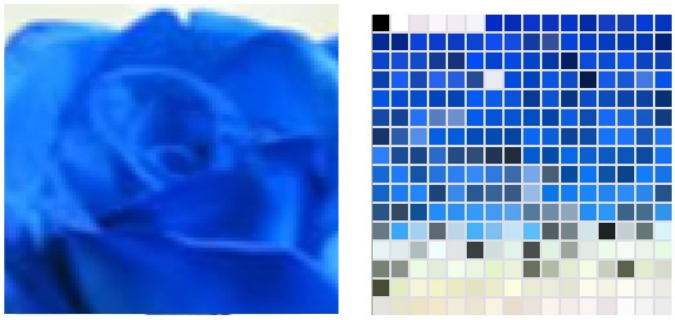
Při aplikaci takové palety na obraz docílíme toho, že každý pixel místo přesné informace o barvě obsahuje pouze index odkazující na barvu v paletě. To znamená,

2 Zdroj: http://www.blueroseflorist.com/images/blue_rose_exclusives.jpg

že se tím zásadně sníží velikost obrazu, ale barevný prostor bude omezen pouze na 256 hodnot. Při použití standardní palety ovšem může nastat situace, kdy je pro obraz použitelný pouze malý počet barev z palety, protože paleta obsahuje barvy (odstíny barev), které se v obraze vůbec nevyskytují. Při takové aplikaci palety zpravidla dochází ke značné ztrátě kvality (Obr. 3). Tento jev redukuje vytvoření a použití palety adaptivní.



Obr. 3: Výřez z originálu po aplikaci standardní palety



Obr. 4: Výřez z originálu po aplikaci adaptivní palety

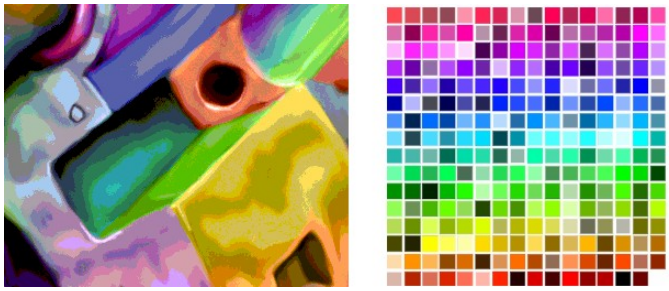
Adaptivní paleta, stejně jako standardní, obsahuje 256 barev, ale nepoužívá rovnoměrné zastoupení barev z celého RGB prostoru. Nejprve se provede analýza obrazu, potom se vyhodnotí, které barvy jsou v obraze nejvíce zastoupeny a z výsledku se vypočítá 256 barev pro paletu (Obr. 4). Tato paleta většinou lépe zastupuje barvy, které se v obrazu skutečně vyskytují.

Případ, kdy tomu tak není, nastane tehdy, když obraz, pro který je paleta vytvářena, obsahuje tolik barev a jejich odstínů (Obr. 5)[3], že představuje téměř celé barevné spektrum. Potom se složení adaptivní palety může velmi podobat standardní paletě a mizí význam takovou paletu vytvářet.

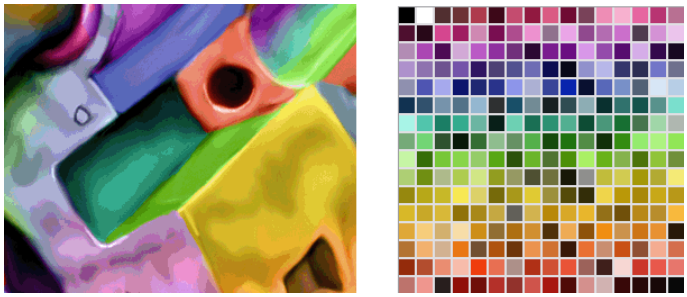
3 Zdroj: http://www.chrispass.com/wp-content/uploads/2006/03/psycheabstraction12809601.jpg

Obr. 5: Originál obrazu, kde nemá smysl vytvářet adaptivní paletu

Výsledek po aplikaci adaptivní palety (Obr. 6) je srovnatelný s použitím standardní palety (Obr. 7). V tomto případě ji jsou palety velice podobné.



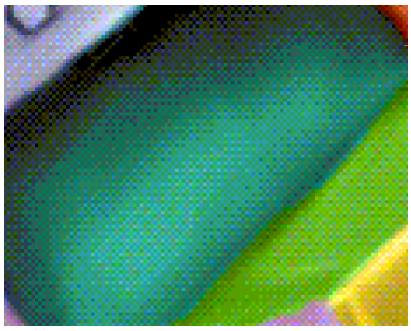
Obr. 6: Výřez z originálu po aplikaci standardní palety



Obr. 7: Výřez z originálu po aplikaci adaptivní palety

Paleta (standardní i adaptivní) může být použita ve všech formátech, které ji podporují. Typickým příkladem je formát GIF.

Při aplikaci palety na obraz bývá často využit tzv. dithering (Obr. 8). Jedná se o vyplňování vzorem, který je složen z dvou a více barev. Ve výsledku se pak přechod mezi barvami zdá opticky plynulejší, než při nahrazení originální barvy nejbližší barvou z palety.



Obr. 8: Ukázka ditheringu

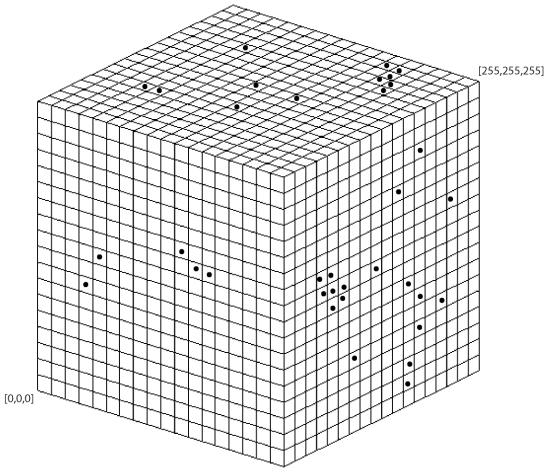
2.2 Návrh řešení

V celém algoritmu jde o to, vybrat co nejvhodněji 256 oblastí; z těchto kvádrových oblastí se pak na základě jedné z možných metod vybere jedna barva, která by měla co nejvhodněji zastupovat všechny barvy v oblasti. Při zmenšování oblastí může nastat situace, kdy najdeme četnost např. s hodnotou 1. Otázkou zůstává, zda takovou hodnotu zanedbat, nebo ji zahrnout do konečného zpracování. Přesná odpověď na tuto otázku není. V případě, kdy obraz obsahuje např. 200 barev o velkých četnostech a 100 barev o potencionálně zanedbatelných četnostech, pak se tyto barvy do palety nedostanou a na výsledném obrazu tvořeném takovou paletou se to velice znatelně projeví. Pokud je obraz tvořen velkým množstvím barev s relativně rovnoměrně rozloženými četnostmi, pak se vynechání barev s potencionálně zanedbatelnou četností ve výsledku nijak viditelně neprojeví. Rozlišování v jakém případě je vhodná jaká tolerance není jednoduché, proto se při dalším postupu bude brát jako platná i četnost o hodnotě 1.

2.2.1 3D histogram

Nejprve je potřeba vytvořit histogram barev v obrazu; ovšem nejedná se o běžný 2D histogram, popisující jasový charakter obrazu, ale trojrozměrný histogram, který bude mít tvar krychle o hraně 256, bude tedy reprezentovat celý RGB barevný prostor, do něhož se ukládají četnosti barev (Obr. 26).

Dále se dá postupovat dvěma způsoby; „zdola nahoru“, což v praxi znamená shlukové metody, a efektivnější způsob „shora dolů“, který v praxi reprezentuje metoda zvaná *zmenši a rozděl (shrink & split)*. Více o této metodě v [1].

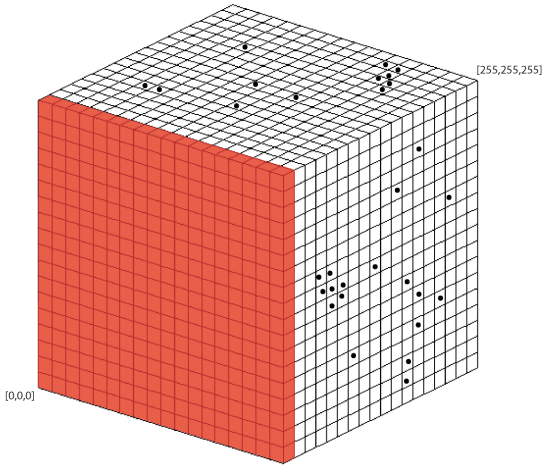


Obr. 9: Příklad 3D histogramu – tečka představuje nenulovou četnost

Pokud máme barevný prostor s histogramem barev, bude třeba v něm najít konečný počet oblastí (v tomto případě 256), po kterém můžeme říct, že každá tato oblast bude v paletě zastoupena jednou konkrétní barvou. Před dalším postupem bude třeba objasnit, jak funguje algoritmus pro zmenšování oblastí v tomto prostoru.

2.2.2 Algoritmus zmenšení prostoru

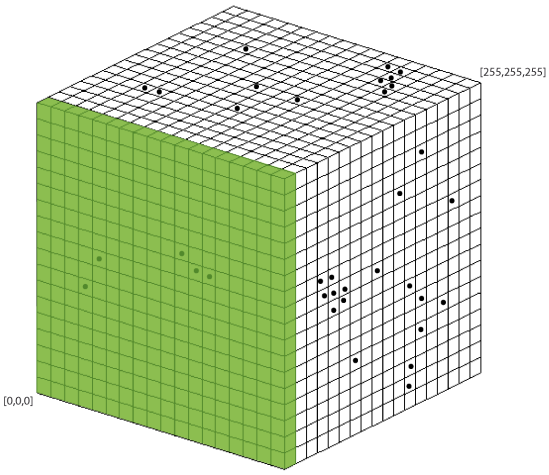
Představme si, že prostor je tvořen z jednotlivých slupek o tloušťce 1 voxel[4]. Pak budeme z každé strany kvádru brát vždy jednu vrstvu voxelů a zkoumat, zda se na této vrstvě vyskytuje barva, která má v obrazu nenulovou četnost. Pokud zastoupena není (Obr. 10), nebudeme dál tuto vrstvu brát v úvahu (zmenšíme tedy zkoumanou stranu kvádru o jednu slupku) a stejným způsobem analyzujeme následující vrstvu strany,



Obr. 10: Slupka obsahuje pouze nulové četnosti – dále se s ní nepočítá

4 Volume element – nejmenší stavební prvek v 3D prostoru. Analogie k 2D základnímu prvku pixel.

dokud nenarazíme na vrstvu v které se vyskytuje nenulová četnost (Obr. 11). Pak kvádr z této strany prohlásíme za dostatečně zmenšenou. Analogicky ošetříme zbylé strany kvádru.

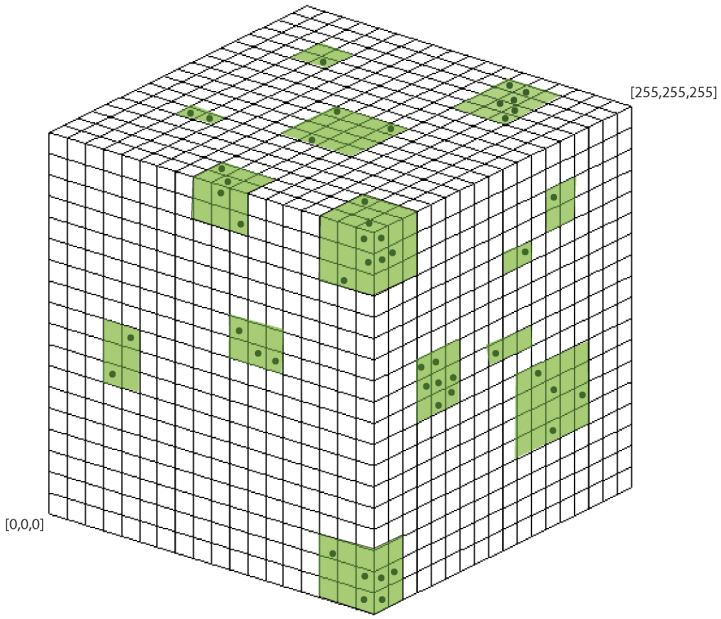


Obr. 11: Slupka obsahuje nenulové četnosti – bude dál zpracována

2.2.3 Výpočet oblastí

Na počátku je RGB krychle naplněná četnostmi barev z obrazu. Před prvním dělením prostoru se krychle ořeže o prázdné vrstvy pomocí algoritmu na zmenšení prostoru. Tím získáme první oblast. Otázkou zůstává, jak tuto oblast vhodně rozdělit. Nejjednodušší způsob je vybrat nejdelší hranu, podle které se krychle rozdělí na dvě symetrické části. Efektivnější by ovšem bylo najít takovou řeznou rovinu, která prostor rozdělí na dvě části obsahující stejný počet barev. A nejlepší řešení je rozdělit prostor tak, aby se součet četností jednoho prostoru rovnal součtu četností druhého prostoru. V implementaci je použitá první metoda, dělení na půlky podle nejdelší hrany.

Po rozdělení na dvě poloviny má každá, ze dvou nově vzniklých oblastí jednu stranu (tu, co vznikla po rozpůlení), která není ošetřena algoritmem pro zmenšení oblasti. Tyto strany se zmenší a celý postup se znovu opakuje, dokud není dosaženo počtu 256 oblastí. Při vybírání vhodné oblasti pro rozdělení je důležité vždy zvolit takovou oblast, která má nejdelší hranu a podle této hrany dělit. Tím se zamezí vzniku příliš „dlouhých“ oblastí, které by mohly obsahovat hodně barevných odstínů a barva, která takovou oblast bude zastupovat nenahradí vhodně celé spektrum oblasti. Proto je algoritmus navržený tak, aby maximalizoval minimální délku hran (Obr. 12).



Obr. 12: Náhled na krychli s vypočtenými oblastmi (oblasti zasahují a nacházejí se i uvnitř krychle)

2.2.4 Vytvoření palety

Z každé oblasti je nyní třeba vhodně vypočítat barvu, která ji bude zastupovat. Jako jeden z možných způsobů výpočtu byl zvolen postup pomocí klouzavého průměru (Vzorec 2).

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i \cdot x_i}{\sum_{i=1}^{n} w_i}; X = \{x_1, x_2 \ldots, x_n\}, W = \{w_1, w_2 \ldots, w_n\}$$

Vzorec 2

Ve vzorci představuje X_i soubor hodnot a W_i jemu odpovídající váhu. Pro příklad poslouží barvy a jejich četnosti zapsané v tabulce (Tabulka 1).

Barva	***Červená (R)***	***Zelená (G)***	***Modrá (B)***	***Četnost***
1.	56	34	134	100
2.	58	30	140	50
3.	50	31	139	1

Tabulka 1

Výsledná barva se vypočítá takto:

$$R = \frac{100 \cdot 56 + 50 \cdot 58 + 1 \cdot 50}{100 + 50 + 1} \approx 56$$
$$G = \frac{100 \cdot 34 + 50 \cdot 30 + 1 \cdot 31}{100 + 50 + 1} \approx 33$$
$$B = \frac{100 \cdot 134 + 50 \cdot 140 + 1 \cdot 139}{100 + 50 + 1} \approx 136$$

Vypočítaná zástupná barva bude mít RGB hodnoty [56, 33, 136]. To znamená, že každá barva se ve výsledku projeví podle své „váhy“, v tomto případě podle toho, jaký podíl má na celkové četnosti v oblasti. Vypočítané barvy se převedou do palety a aplikují se na obraz.

2.3 Implementace

2.3.1 TObalka

3D barevný RGB prostor (obálka) je definován v třídě *TObalka*. Datová struktura obálky je řešená jako statické trojrozměrné pole datového typu *integer*.

```
mObalka: array[0..255,0..255,0..255] of integer;
```

Protože tato struktura zabírá 64 MB, je vhodné ji v programu vytvořit pouze jednou a dále ji mezi třídami přenášet odkazem. Při zvolení menšího datového typu hrozí, že četnost přesáhne maximální hranici datového typu. Při zvolení typu *integer*[5] se může ve zpracovávaném obraze objevit přes 2 miliardy pixelů se stejnou barevnou hodnotou. Při použití dvoubajtové proměnné je maximum 64 565, což v některých případech nemusí stačit. Díky tomu, že se jedná o statickou strukturu s přímým přístupem přes indexy, lze s ní velmi rychle pracovat i přes její datový objem.

Při vytváření histogramu se může celý průběh velice znatelně zpomalit, pokud se použije přístup k pixelům v obrazu přes indexy. Tento způsob je implementován ve třídě *TZpracovani*.

```
procedure VytvorObalku(Obraz: TBitmap);
```

Jádro metody:

```
for i:=0 to Obraz.Height do
  for j:=0 to Obraz.Width do
  begin
```

5 Nyní datový typ integer zabírá 4 bajty.

```
      Barva:=Obraz.Canvas.Pixels[i,j];
      r:=GetRValue(Barva);
      g:=GetGValue(Barva);
      b:=GetBValue(Barva);
      mObalka.Inkrementuj(r,g,b);
    end;
```

Pro odstranění tohoto prodlení byla napsána alternativní metoda v třídě *TZpracování*.

```
procedure VytvorObalkuRychle(Obraz: TBitmap);
```

Tato metoda využívá k pixelům přímý přístup do paměti. Důležitou roli zde hraje následující lokální proměnná.

```
Radek:PRGBTriple;
```

Je to ukazatel do paměti, který přes své atributy *rgbtRed*, *rgbtGreen* a *rgbtBlue* dovede z paměti odečíst hodnoty jednotlivých barevných složek pixelu. Inkrementací proměnné *Radek,* se posune ukazatel o jeden pixel dopředu, tím je možné se lehce pohybovat po obrazu v horizontálním směru. Pro vertikální se používá metoda *ScanLine*, která vrátí ukazatel na první pixel na řádku *i*.

```
for i:=1 to Obraz.Height-1 do
begin
  Radek:=Obraz.ScanLine[i];
  for j:=0 to Obraz.Width do
  begin
    r:=Radek^.rgbtRed;
    g:=Radek^.rgbtGreen;
    b:=Radek^.rgbtBlue;
    mObalka.Inkrementuj(r,g,b);
    Inc(Radek);
  end;
end;
```

Při použití této metody se celý algoritmus velice znatelně zrychlí a dosáhne se stejných výsledků, jako při použití přístupu k pixelům přes indexy.

2.3.2 TOblast

Tato třída reprezentuje dvě 3D souřadnice, které odkazují do RGB krychle a tím vymezují kvádrovou oblast (souřadnice představují dva rohy oblasti, položené úh-

lopříčně proti sobě), ze které se na konci algoritmu vypočítá barva, která tuto oblast zastoupí.

```
mR:array[1..2] of Integer;
mG:array[1..2] of Integer;
mB:array[1..2] of Integer;
```

Mezi metody oblasti patří její zmenšení o nepotřebné vrstvy. K tomu slouží tři metody, každá pro zmenšení podle jedné z RGB os, což umožňuje zmenšovat jen podle té osy, podle které je v danou situaci potřeba. V oblasti platí pravidlo, že souřadnice indexem 1 mají nižší hodnoty, než souřadnice s indexem 2. Pokud je toto pravidlo porušeno, funkce vrátí *false* a oblast je neplatná (tzn. nebyly v ní nalezeny žádné nezanedbatelné hodnoty). Jedná se o nejsložitější funkci v algoritmu, ale jejich provedení je rychlé.

Oblast též vrací hodnotu své nejdelší hrany. Podle toho se pak vybere vhodná oblast pro rozdělení. Půlení oblasti je implementováno také v této třídě a její návratová hodnota je nová oblast typu *TOblast*. Dále je v třídě též implementován algoritmus na vypočítání zástupné barvy pomocí váženého průměru.

2.3.3 TZpracovani

Hlavním atributem této třídy je pole oblastí, které se vypočítají z RGB krychle.

```
mOblasti: array[0..255] of TOblast;
```

Třída představuje rozhraní mezi formulářem a datovou strukturou programu.

Nejprve třída vytvoří a zinicializuje všechny potřebné datové struktury pro vytváření palety. Po inicializaci naplní RGB krychli četnostmi, nastaví první oblast na celý prostor a provádí zmenšování a dělení oblastí tak dlouho, dokud se nedosáhne konečného počtu. Před vytvořením palety se vypočítané barvy nejprve uloží do pole, pomocí kterého se vykreslí paleta na bitmapu a pošle se jako návratová hodnota do formuláře, kde je zobrazena uživateli.

Pro převedení obrazu na režim 256 barev je nejprve nutné nastavit vlastnost *PixelFormat* bitmapy na hodnotu *pf8bit*, což znamená barevnou hloubku 8 bitů a použití palety.

Samotné vytvoření palety se provede pomocí pomocné struktury.

```
LogPal = record
  Pal : TLogPalette;
  Alokace:Array[0..255] of TPaletteEntry;
end;
```

Atribut *Pal* představuje paletu o následujících vlastnostech.

```
palVersion: Word;
palNumEntries: Word;
palPalEntry: array [0..0] of tagPALETTEENTRY;
```

Atribut *palVersion* je typ palety zadávaný v hexadecimální podobě, *palNumEntries* je počet barev, který bude paleta obsahovat a *palPalEntry* je pole barev palety. To má ovšem rozměr 0. Proto má záznam *LogPal* ještě jeden atribut *Alokace*, který se nepoužívá, ale slouží jako alokování potřebného místa pro *palPalEntry*. Díky tomu je možné dynamicky nastavit velikost statického pole.

2.3.4 UML diagram tříd

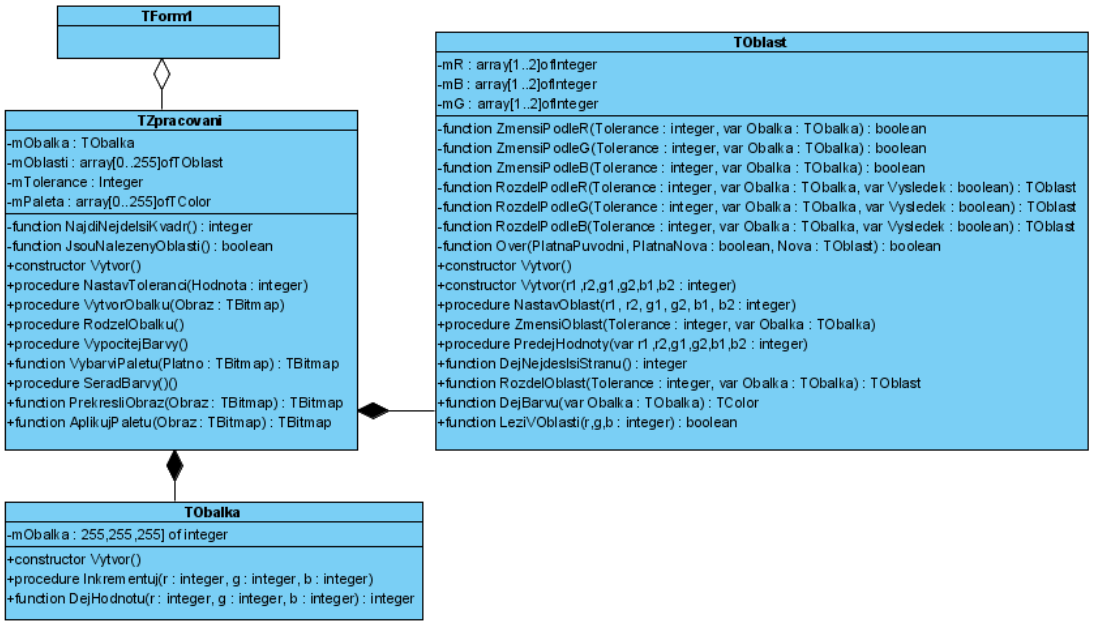


Diagram je ve větší podobě v programátorské dokumentaci, kde jsou i popisky k jednotlivým atributům a metodám.

2.4 Dosažené výsledky

Celé zpracování nejvíce ztrácelo na vytváření histogramu, kvůli přístupu k pixelům přes indexy. Po zrychlení tohoto problému pomocí přímého přístupu do paměti, se algo-

ritmus znatelně zrychlil a jeho celé provedení trvá řádově v jednotkách sekund (většinou se pohybuje kolem dvou sekund).

V následující tabulce (Tabulka 2) je vidět doba trvání[6] jednotlivých operací. Při měření se algoritmus použil na obraz o rozlišení 1600 × 1200 a barevnou hloubkou 24 bitů.

Část algoritmu	***Čas [s]***	***Výpočetní složitost***
Inicializace	0,25	$\Theta(1)$
Vytvoření histogramu	0,16	$\Theta(N^2)$
Výpočet oblastí	0,28	$\Theta(1)$
Výpočet barev z oblastí	0,3	$\Theta(1)$
Vybarvení palety (nemusí proběhnout)	0,02	$\Theta(1)$
Aplikace palety	0,02	$\Theta(1)$
Celkem	**0,73**[6]	**$\Theta(1)$**

Tabulka 2: doba trvání jednotlivých částí; průměr ze 100 provedených cyklů

V algoritmu se vyskytuje pouze jediná část, která má složitost $\Theta(N^2)$, ale tato část zabírá v poměru k celkovému času zpracování zanedbatelnou část (cca 11% z 0,75 sekundy, takže celková složitost se může považovat $\Theta(1)$. Tím se dá považovat algoritmus pro potřeby aplikace jako dostatečně odladěný.

Pro ještě větší zrychlení je možné řadit oblasti podle nejdelší hrany; při hledání vhodné oblasti pro dělení potom nebude třeba prohledávat všechny oblasti. Ovšem pro správnou funkci je vhodné zaměnit strukturu statického pole za jinou strukturu, např. lineární seznam.

Další alternativou k adaptivní paletě může být např. perceptualní paleta. Tato paleta se zaměřuje především na barvy, na které jsou v obrazu nejvíce zastoupeny a lidské oko je na ně nejcitlivější. Další možnou alternativou je selektivní paleta. Ta vygeneruje podobné barevné spektrum jako perceptuální paleta, ale více upřednostňuje souvislé barevné plochy.

6 Testováno na počítači Acer Aspire 3693, procesor Intel Celeron 1.6 GHz, 760 MB operační paměti, operační systém Windows XP®.

3 Spojení obrazů

3.1 Teoretický rozbor

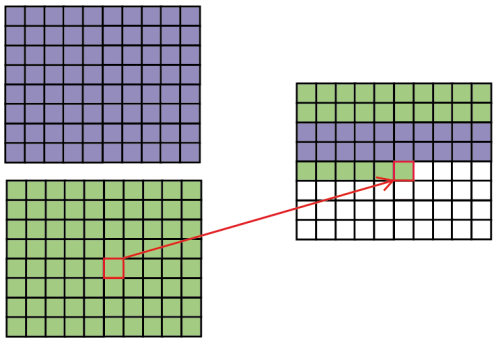
Hlavní činnost aplikace spočívá ve specifickém spojení obrazů. Cílem této části je navrhnout a implementovat takový algoritmus, který toto spojení vykoná v přijatelném čase. Tzn. navrhnout několik variant a vybrat nejvhodnější řešení.

Výsledkem tohoto spojení je obraz, který je složený z proužků složených střídavě z prvního a druhého obrazu. Počet řádků, které budou tvořit jeden proužek je nastavitelný uživatelem. Takto spojený obraz potom bude fungovat jako reklamní plocha. Více v kapitole 6.

3.2 Návrh řešení

3.2.1 Kopírování po pixelech

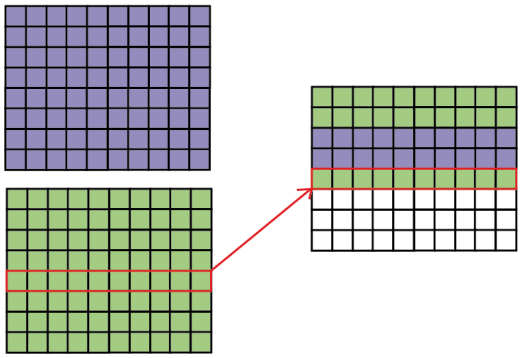
Nejjednodušší řešení je spojovat obrazy pixel po pixelu; to znamená nahrát do programu oba vstupní obrazy, připravit prostředí pro obraz výstupní a pak kopírovat pixel za pixelem z jednoho, nebo z druhého obrazu (Obr. 13). Pro to, z kterého obrazu se bude kopírovat, je třeba určit nějaké kritérium. Jeden ze způsobů je použití bezezbytkového dělení (více v části 3.3.1).



Obr. 13: Kopírování po pixelech

3.2.2 Kopírování po řádcích

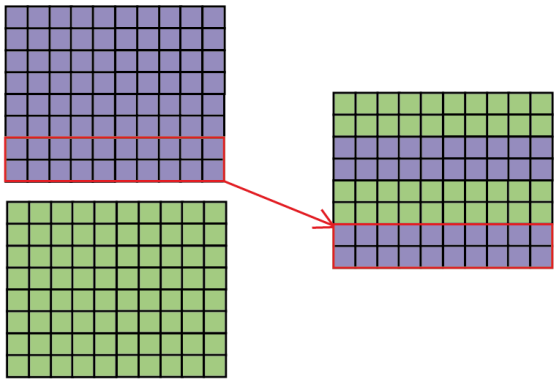
Metoda pixel po pixelu není pro tento případ moc efektivní, protože je předem známé, že se budou kopírovat celé řádky bez přerušení, tudíž bude efektivnější kopírovat celé řádky. Borlad Delphi nabízí metodu *ScanLine* (Obr. 14), která vrací směrník do paměti na adresu řádku. To poskytuje možnost přímé, nezprostředkované práce s pamětí, což může celý algoritmus ještě urychlit.



Obr. 14: Kopírování pomocí ScanLine

3.2.3 Kopírování datových bloků

Protože šířka kopírovaného pruhu může být větší jak 1, znamená to, že je předem známé, že se bude kopírovat více řádků z jednoho obrazu za sebou. Zde se nabízí možnost zkopírovat celý blok dat představující tyto řádky (Obr. 15). Tento způsob zprostředkuje v Borland Delphi metoda *CopyRect*.



Obr. 15: Kopírování pomocí CopyRect

3.3 Implementace

Celá implementace je napsaná v modulu *uSpojení* ve třídě *TSpojení*. Disponuje dvěma atributy.

```
mTloustka:Integer;
mVyslednyObraz:TBitmap;
```

Atribut *mTloustka* představuje počet řádků, které mají reprezentovat jeden pruh ve výsledném obrazu, který je reprezentován v atributu *mVyslednyObraz*. Společné pro všechny druhy spojení jsou tyto metody.

```
procedure NastavTloustku(Tloustka:Integer);
function VratVysledek():TBitmap;
constructor Vytvor(Sirka, Vyska:Integer);
destructor Zrus();
```

Metody slouží pro nastavení atributu *mTloustka*, a pro vrácení atributu *mVyslednyObraz* po zavolání některé metody pro spojení. Konstruktor zinicializuje atribut *mVyslednyObraz* a nastaví jeho šířku a výšku na vstupní hodnoty. Tyto hodnoty se mohou vzít jak z prvního, jak z druhého obrazu, protože v této části programu se již počítá s tím, že obrazy mají shodné rozměry, jelikož je to nutná podmínka pro úspěšné spojení. Destruktor se postará o to, aby se paměť alokovaná pro potřeby spojení uvolnila. Tyto vlastnosti třídy fungují stejně, nezávisle na zvolené metodě spojení.

3.3.1 Kopírování po pixelech

Metoda je implementována v proceduře *SpojPoPixelech*.

```
procedure SpojPoPixelech(Obraz1:TBitmap; Obraz2:TBitmap);
```

Jako vstupní parametry má bitmapy obrazů určených se spojení. K řízení spojení slouží lokální proměnné.

```
i,j,Pruh:integer
```

Ty slouží jako počítadla šířky a výšky a proměnná *Pruh* slouží k počítání, kolikátý pruh se právě kopíruje; tato proměnná slouží k rozlišení sudého a lichého pruhu, tím se rozpozná, ze kterého obrazu se má právě kopírovat. V programu toto kriterium vypadá následovně:

```
Pruh:=0;
for i:=0 to Obraz1.Height do
begin
  if((i mod mTloustka)=0)then
    inc(Pruh);
  if((Pruh mod 2)=0)then
    for j:=0 to Obraz1.Width do
      mVyslednyObraz.Canvas.Pixels[j,i]:=Obraz1.Canvas.Pixels[j,i];
  if((Pruh mod 2)<>0)then
    for j:=0 to Obraz1.Width do
      mVyslednyObraz.Canvas.Pixels[j,i]:=Obraz2.Canvas.Pixels[j,i];
end;
```

V proceduře hraje hlavní roli operace bezezbytkového dělení *mod*. V prvním příkazu *if* se rozpozná, zda byl překopírován celý pruh, pokud ano, proměnná *Pruh* se inkrementuje a tím se cyklicky přepíná mezi sudou a lichou hodnotou, což slouží jako kriterium pro to, z kterého obrazu se má kopírovat.

3.3.2 Kopírování po řádcích

Metoda je implementována v proceduře *SpojPomociScanLine*.

```
procedure SpojPomociScanLine(Obraz1,Obraz2:TBitmap; PF:TPixelFormat);
```

Zde se jako parametry předávají obrazy určené ke spojení a vlastnost *PixelFormat*, která slouží pro nastavení barevné hloubky výsledného obrazu. Bez nastavení atributu *PixelFormat*, nelze spojení pomocí této metody realizovat, proto je nutné, aby byl nastaven, nejlépe na hodnotu barevné hloubky obrazů, které se spojují.

Před samotným spojováním je potřeba znát datový objem jednoho řádku, který se vypočítá takto:

```
Sirka:=Obraz1.Width * 3;
```

Zde se provádí násobení hodnotou 3, protože datový objem jednoho pixelu (popsaném v datovém typu *TRGBTriple*) nabývá hodnoty 3 bajty. Tím se vypočítá, jak velký je jeden řádek obrazu v bajtech. Pokud je známa tato hodnota, provede se jádro procedury.

```
for i:=0 to (Obraz1.Height-1) do
begin
  Cislo:=i;
  if((Cislo mod mTloustka)=0)then
    inc(Pruh);
  if((Pruh mod 2)=0)then
  begin
    Linka1:=Obraz1.ScanLine[i];
    Linka2:=mVyslednyObraz.ScanLine[i];
    move(Linka1^,Linka2^,Sirka);
  end;
  if((Pruh mod 2)<>0)then
  begin
    Linka1:=Obraz2.ScanLine[i];
    Linka2:=mVyslednyObraz.ScanLine[i];
    move(Linka1^,Linka2^,Sirka);
  end;
end;
```

Kriterium pro přepínání je stejné jako u metody *Kopírování po pixelech*. Samotné přesunutí řádku proběhne tak, že se nejdříve provede operace *ScanLine*, která vrátí ukazatel na místo v paměti, kde je zapsán řádek obrazu na indexu *i*. Tímto způsobem se získají ukazatele na řádek v obrazu, ze kterého se bude kopírovat a řádek obrazu,

do kterého se bude kopírovat. Ukazatele se uloží do proměnných *Linka1*, *Linka2* a potom se s nimi provede operace *move*, která přesune z místa *Linka1* data, o délce *Sirka* (vypočtená datová délka řádku z předchozího kroku) na místo, kam ukazuje *Linka2*. Tím se provede kopie celého řádku z originálu do výsledného obrazu. Taková operace je mnohem rychlejší díky přímému přístupu do paměti a také proto, že odpadne práce s každým pixelem jednotlivě.

3.3.3 Kopírování datových bloků

Metoda je implementována v proceduře *SpojPomociCopyRect*.

```
procedure SpojPomociCopyRect(Obraz1,Obraz2:TBitmap);
```

Zde se stejně jako u předchozích metod, obrazy určené ke spojení předávají pomocí parametrů. Protože se zde budou kopírovat celé pruhy složené z jednoho a více řádků obrazu, může se použít jednodušší kriterium pro přepínání zdrojových obrazů. Jelikož se bude přepínat po každém cyklu, stačí se řídit tím, zda je aktuální číslo cyklu sudé, či liché. Počet cyklů se jednoduše vypočítá poměrem výšky obrazu a tloušťky pruhu (v pixelech) a je uložen do proměnné *Pocet*. Další důležitou proměnnou v této metodě je *Obdelnik*.

```
Obdelnik:TRect;
```

Do této se ukládají souřadnice pravoúhlé oblasti, která se bude kopírovat.

```
Pocet:=round(mVyslednyObraz.Width/mTloustka);
  for i:=0 to Cislo do
  begin
    Obdelnik.Left:=0;
    Obdelnik.Right:=Obraz1.Width;
    Obdelnik.Top:=mTloustka*i;
    Obdelnik.Bottom:=mTloustka*(i+1);
    if (i mod 2)=0 then
    begin
      mVyslednyObraz.Canvas.CopyRect(Obdelnik,Obraz2.Canvas,Obdelnik);
    end
    else
    begin
      mVyslednyObraz.Canvas.CopyRect(Obdelnik,Obraz1.Canvas,Obdelnik);
    end;
  end;
```

Proměnnou *Obdelnik* stačí nastavit při každém cyklu jednou, nezávisle na tom, ze kterého obrazu se bude kopírovat. Jeho rozměry se můžou nastavit podle jednoho i podle druhého obrazu; jedná se o nekonkrétní souřadnice. Po nastavení pravoúhlé oblasti se podle kritéria rozhodne, ze kterého obrazu se bude kopírovat a pak se provede samotné kopírování. Metoda *CopyRect* se zavolá z atributu *Canvas* cílového obrazu a jako parametr se uvádí pravoúhlá oblast typu *TRect* a *Canvas* zdrojového obrazu.

3.4 Dosažené výsledky

Rychlost zpracování vcelku odpovídá předpokladům. Kopírování po pixelech je zdlouhavé a na jeho provedení se musí čekat řádově v sekundách (až v desítkách sekund), podle toho, jak jsou spojované obrazy velké.

Za použití metody *ScanLine* proběhlo spojení mnohem rychleji, doba trvání je menší než půl sekundy, což je okamžik, který časově předčí samotné zobrazení výsledku, takže samotné spojení uživatel časově ani nezaregistruje.

U metody *CopyRect* byl původní předpoklad, že by mohla být ještě rychlejší, než *ScanLine*. Ve výsledku ovšem spojení tímto způsobem trvá zhruba stejně dlouho jako u metody *ScanLine*, čas ušetřený tímto způsobem je pro lidské vnímání naprosto zanedbatelný, takže samotné spojení jako v předchozím případě uživatel ani nezaregistruje.

V tabulce (Tabulka 3) jsou uvedeny časy[6] a složitosti metod. Aplikace se testovala pro spojení obrazů o formátu A1 a rozlišení 150 DPI (což představuje 4967 × 3068 pixelů), protože přibližně takové obrazy se budou v praxi spojovat.

Metoda	***Čas zpracování [s]***	***Složitost výpočtu***
Po pixelech	39	$\Theta(N^2)$
ScanLine	0,4	$\Theta(N^2)$
CopyRect	0,1	$\Theta(N^2)$

Tabulka 3

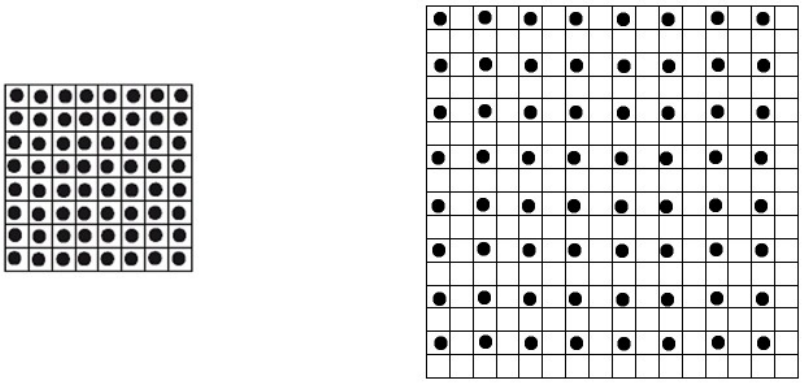
Závěrem lze prohlásit, že v těchto možnostech spojení je důležitý pro rychlost způsob, jakým se k datům přistupuje. Přístup přes indexy je v měřítku, v jakém je použitý zde, značně neefektivní. Další dvě metody používají přístup přímo do paměti, což je pro odstranění tohoto problému klíčové. S tímto přístupem se dá dosáhnout srovnatelných výsledků i přes to, že by měla být metoda při použití *CopyRect* rychlejší. Za použití jedné ze dvou metod, které pracují přímo s pamětí, se může prohlásit, že se dosáhlo uspokojivých výsledků.

4 Interpolace

4.1 Teoretický rozbor

Interpolace, ve volném překladu, znamená doplňování hodnot do intervalu. Jakým způsobem se tyto hodnoty vypočítají, je závislé na konkrétním případě, kdy je interpolace použita a na metodě výpočtu.

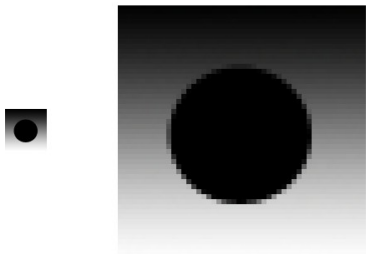
V rastrové grafice interpolace znamená změnu velikosti obrazu. Kdyby nebyla použita, a zvětšoval by se např. obrázek o velikosti 8×8 pixelů na velikost 16×16, vznikly by na rastru prázdné pixely, tzv. „díry“ (Obr. 16). Pro odstranění těchto děr, resp. jejich zaplnění se musí použít nějaká metoda interpolace.



Obr. 16: Převedení obrazu 8×8 na 16×16 a následný vznik děr (bílá pole)

4.1.1 Interpolace nejbližším sousedem

Jedná se o nejjednodušší a nejrychlejší způsob interpolace, nazývaný též *point shift*, nebo také *sample and hold* (Obr. 17). Ze souřadnic pixelu, který má být právě obarven, se vypočítá jeho nejbližší soused z originálu a obarví se na jeho hodnotu. Jednoduchá



Obr. 17: Zvětšení metodou nejbližšího souseda

a rychlá implementace se ovšem projeví na výsledku nežádoucími jevy, jako je zvýraznění jasových (barevných) skoků, ztráta celistvosti hran s malým sklonem (obraz se pak vypadá jako mozaika složená ze čtverců), naopak při zmenšování můžou být velmi poškozeny, nebo úplně ztraceny tenké čáry. V praxi je vhodné tuto metodu používat především pro náhledové zvětšení, či zmenšení a to díky své rychlosti. Tuto metodu interpolace používá standardně metoda *StretchDraw* v prostředí Borland Delphi, což dává impuls k tomu, aby byla napsána důmyslnější metoda interpolace.

4.1.2 Lineární a bilineární interpolace

Tento způsob interpolace už je výpočetně náročnější, ale výsledek je zcela odlišný od předchozí metody (Obr. 18). Lineární interpolace při výpočtu barev mezi původními pixely již uvažuje se změnou barevné hodnoty v závislosti na změně mezi pixely, mezi kterými se barvy dopočítávají. Když má tedy pixel A na souřadnici o barevnou hodnotu X_A a pixel B barevnou hodnotu X_B a po zvětšení bude třeba dopočítat n pixelů mezi těmito původními, bude jejich barevná informace nabývat hodnot v intervalu $<X_A, X_B>$. Tyto hodnoty se vypočítají tak, že se body A a B proloží přímkou a z hodnot, které bude přímka nabývat v požadovaném bodě se odečte hodnota barvy nového pixelu.

Bilineární interpolace se docílí tak, že se provede lineární interpolace v obou směrech, tzn. podle osy x a y.

Obr. 18: Zvetšení bilineární metodou

4.1.3 Kubická a bikubická interpolace

Pokud je požadavek na ještě lepší vyhlazení zvětšeného originálu, jednou z dalších možností je kubická interpolace. To znamená, že průběh funkce mezi dvěma pixely není znázorněn přímkou, ale funkcí složenou z kubických křivek[7], která bude více respektovat přechody mezi barvami. Aby mohla být taková funkce vytvořena, je potřeba znát

7 Kubická křivka – křivka třetího řádu, tzn. je určena čtyřmi body

více okolních bodů pixelu, což znamená, že taková interpolace bude výpočetně náročnější než předchozí metody. Více o bikubické interpolaci v [1].

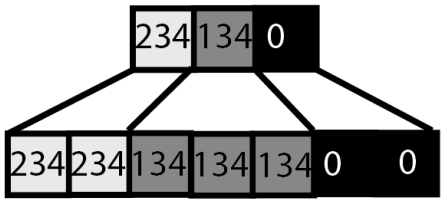
4.2 Návrh řešení

4.2.1 Interpolace nejbližším sousedem

Jedním ze způsobů je nalézání nejbližšího souseda podle poměru stran. Souřadnice pixelu, který je potřeba vypočítat se vydělí poměrem (vertikálním pro osu *y*, horizontálním pro osu *x*) a po zaokrouhlení reálných čísel je vypočtená souřadnice pixelu v originálním obrazu, ze kterého se zkopíruje jeho barevná informace do interpolovaného obrazu. Zde záleží tom, jak se poměr stran vypočítá (Vzorec 3).

$$Poměr = \frac{Originál}{Interpolovaný} \Rightarrow Nový\ pixel = [x, y] \cdot Poměr$$
$$Poměr = \frac{Interpolovaný}{Originál} \Rightarrow Nový\ pixel = \frac{[x, y]}{Poměr}$$

Vzorec 3



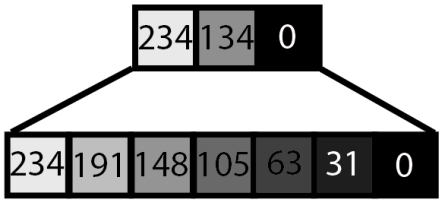
Obr. 19: Příklad interpolace nejbližším sousedem

4.2.2 Lineární a bilineární interpolace

Bilineární interpolace se dá implementovat dvěma způsoby. Buď se provede nejprve interpolace ve směru jedné osy a pak se směru druhé, nebo se při výpočtu uvažují oba směry najednou. V obou metodách jde vlastně v zásadě o nalezení okolních pixelů z originálu a vypočtení, na jaké pozici se tento pixel nachází pro odečtení jeho hodnoty z lineární funkce (Vzorec 4).

$$f(x) = f_0 + [\frac{x - x_0}{x_1 - x_0} \cdot (f_1 - f_0)]$$

Vzorec 4



Obr. 20: Příklad lineární interpolace

4.3 Implementace

Metody pro interpolaci jsou implementovány ve třídě *TInterpolace* v modulu *uInterpolace*. Má pouze jeden atribut a to je bitmapa, kam se ukládá výsledný obraz.

```
mVysledek:TBitmap;
```

Pro všechny metody interpolace je společný konstruktor a destruktor.

```
constructor Vytvor(Sirka,Vyska:Integer);
destructor Znic();
```

Konstruktor vytvoří atribut *mVysledek* a nastaví jeho výšku a šířku na vstupní parametry. Ostatní metody jsou popsány níže.

4.3.1 Interpolace nejbližším sousedem

Tato metoda je implementována ve funkci *NejblizsiSoused*.

```
function NejblizsiSoused(Original:TBitmap):TBitmap;
```

Nejjednodušší způsob jak najít nejbližší vybarvený pixel v novém obrazu, je vypočítat poměr stran originálu a interpolovaného obrazu a pomocí dělení (může být i násobení, záleží na tom, jak se poměr vypočítá) vypočítat přibližné souřadnice pixelu v originálu. Jádro funkce vypadá následovně:

```
PomerVysky:=Original.Height/mVysledek.Height;
PomerSirky:=Original.Width/mVysledek.Width;
for j:=0 to mVysledek.Height do
  for i:=0 to mVysledek.Width do
  begin
    x:=Round(i*PomerSirky);
    y:=Round(j*PomerVysky);
    mVysledek.Canvas.Pixels[i,j]:=Original.Canvas.Pixels[x,y];
  end;
```

Protože se vypočítávají indexy, je třeba zaokrouhlit výsledek výpočtu.

4.3.2 Lineární a bilineární interpolace

Tato metoda je implementována ve funkci *Bilinearni*.

```
function Bilinearni(Original:TBitmap):TBitmap;
```

Při implementaci byla zvolena metoda postupné interpolace; tedy nejprve interpolace v jednom směru, kde se výsledek ukládá do záložní bitmapy, která se následně interpoluje v druhém směru. Touto kombinací se docílí bilineární interpolace. Při výpočtu barvy aktuálního pixelu je důležité vědět, mezi kterými pixely z originálu se právě počítá, jeden ze způsobů je tento:

```
Zvetseni:=mVysledek.Height/Original.Height;
IndexNizsiho:=Trunc(i/Zvetseni);
IndexVyssiho:=Trunc(i/Zvetseni)+1;
```

Zde *i* představuje index řádku, na kterém se právě počítá a funkce *Trunc* vrátí celou část reálného čísla. Pokud jsou známy tyto dva pixely, může se odečíst jejich barevné hodnoty pomocí následujících funkcí[8]:

```
r1:=GetRValue(Original.Canvas.Pixels[j,IndexNizsiho]);
r2:=GetRValue(Original.Canvas.Pixels[j,IndexVyssiho]);
```

Pokud jsou známy barevné hodnoty pixelů, resp. rozdíl mezi nimi, může proběhnout samotné odečtení hodnoty z lineární funkce, které je implementováno takto:

```
r:=Round(r1+((r2-r1)*Frac(i/Zvetseni)));
```

Funkce *Frac* vrací hodnotu za desetinnou čárkou z poměru indexu aktuálního pixelu a poměru obou obrazů. Tím se vypočítá pozice, ze které se má na lineární funkci odečítat. Tato hodnota se vynásobí barevnou změnou mezi pixely na originálním obrazu a přičte se (v případě, že má lineární funkce klesající charakter, vyjde záporná hodnota a změna se odečítá) k barevné hodnotě z prvního pixelu originálu. Tím se docílí toho, že barevná hodnota mezi interpolovanými pixely bude plynule přecházet. Po vypočtení všech barevných složek nového pixelu se integrují pomocí funkce RGB do typu TColor a přiřadí se na aktuální souřadnice.

```
Zalozni.Canvas.Pixels[j,i]:=RGB(r,g,b);
```

Pokud se provedla interpolace v jednom směru (v tomto případě ve vertikálním), provede se analogicky interpolace v druhém směru, s tím rozdílem, že jako originál se nyní uvažuje výsledek z předešlé interpolace a jako výstup se použije výsledný obraz.

8 Pro názorný příklad je uvedeno počítání pouze pro červenou složku pixelu; zelená a modrá se vypočítají analogicky.

5 Doplňky

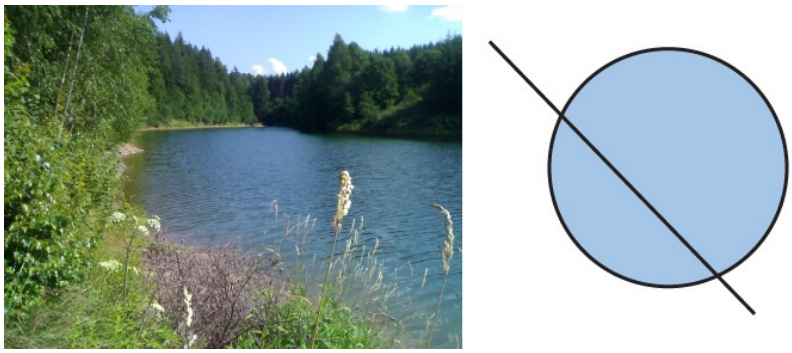
Aby aplikace vyhovovala požadavkům pro praktické využití, je potřeba ji rozšířit o některé doplňky, které standardně vývojové prostředí Borlad Delphi nenabízí. V první řadě je třeba zajistit, aby bylo možné načítat a ukládat více grafických formátů, než pouze BMP. Dále bude potřeba aplikaci rozšířit o možnost nastavení a zjištění DPI obrazu a převod do metrické soustavy, aby bylo možné snadno odečíst, jak velký bude obraz na výstupním zařízení.

5.1 Teoretický rozbor

5.1.1 Grafické formáty

BMP (Microsoft Windows Bitmap)

Historie tohoto formátu sahá do roku 1988, kdy ho představila firma Microsoft. Později byl implementován v operačním systému Microsoft Windows 3.0, který již podporoval 16 bitové grafické rozhraní. Tento formát většinou nepoužívá žádnou kompresi, je tedy často nazýván bitmapovým polem[9]. Běžně pixely mají svou barevnou informaci zapsánu pomocí jednoho bajtu (256 úrovní) na barevnou složku, takže barevná hloubka je 8 × 3 bajty, což je 24 bitů (16 milionů barev). BMP může obsahovat tzv. *Alpha* kanál[10], kde je v jednom bajtu uložena informace o stupni šedé, což zvyšuje barevnou hloubku na 32 bitů. Tento formát může mít i nižší barevnou hloubku, nejnižší 1 bit. Tento formát má také možnost komprese, ale v praxi se tato možnost příliš nevyužívá.



Obr. 21: Originální obrázek ve formátu BMP – 599 kB. Pro příklad je obrázek složen z fotografie a těles s ostrými obrysy

9 Grafická informace je vyjádřena formou matice obrazovkových bodů (pixelů).
10 Tento kanál obsahuje speciální informaci o průhlednosti obrazu. Je definována pomocí stupnice šedi; bílá barva definuje neprůhlednost, černá potom plnou průhlednost.

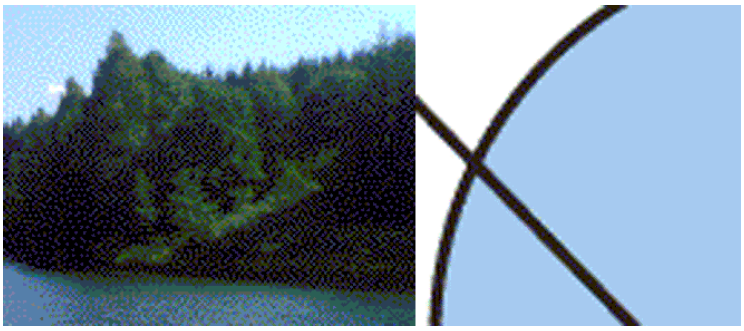
JPEG (Joint Photographic Experts Group)

Jedná se o metodu ztrátové[11] komprese. Samotné soubory používající tuto metodu můžou mít přípony *.jpg, .jpeg, .jfif, .jpe*. Standardizován byl v roce 1991 normou ISO. Skutečný název jednoho z nejrozšířenějších formátů je JFIF, neboli JPEG File Interchange Format. Vhodný pro fotografie obsahující značné množství barev, nevhodný pro plánky, perokresby díky způsobu komprese. Stejně jako BMP má tento formát barevnou hloubku 24 bitů. Vylepšená verze JPEG 2000 byla obohacena o bezztrátovou kompresi. Další vylepšení této verze je ve způsobu komprese; obraz po kompresi novou metodou zabírá méně, než obraz s původní kompresí o stejné kvalitě.

Obr. 22: Výřezy z obrázku ve formátu JPEG – 35 kB, stupeň komprese 12/12

GIF (Graphic Interchange Format)

Formát vyvinutý v roce 1987 firmou CompuServe ve své původní verzi 87a. Dva roky po té přišla druhá verze 80a, která byla oproti původní verzi 97a rozšířena o možnost více obrázků (což umožňuje vytváření jednoduchých animací). Každý rámec



Obr. 23: Výřezy z obrázku ve formátu GIF – 58,7 kB

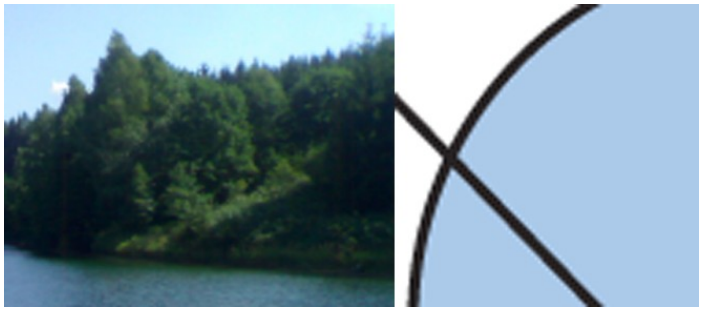
(obrázek, snímek) může mít maximálně 256 (8 bitová barevná hloubka) barev. GIF používá bezztrátovou kompresi LZW841. Je vhodný pro krátké animace, perokresby, plánky, nevhodný pro fotografie. Umožňuje transparentnost (průhlednost) jedné barvy a je vhodný pro webovou grafiku. V roce 2006 vypršela platnost poslední patentové

11 Ztrátová komprese znamená, že obrázek po opakovaném ukládání a otevírání postupně ztrácí na kvalitě podle zvoleného stupně komprese. Naproti tomu bezztrátová komprese sníží kvalitu pouze jednou a ta nadále zůstává stejná i po opakovaném otevírání a ukládání.

ochrany na tento formát, ale to už byl na světě formát, který byl vyvíjen jako nástupce GIFu, formát PNG.

PNG (Portable Network Graphics)

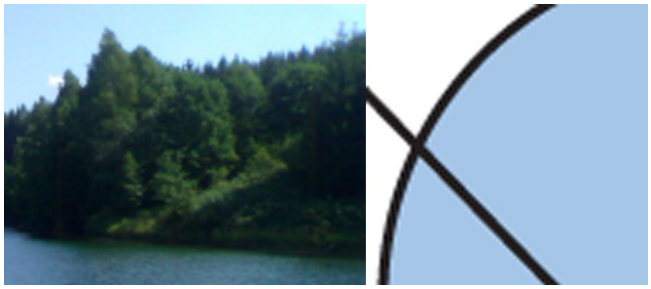
Byl vyvinut na základě bezztrátového kompresního algoritmu *deflation*, kterou používá např. i metoda komprese ZIP. Podnětem k jeho vzniku byla právě patentová ochrana formátu GIF. Umožňuje až 48 bitovou barevnou hloubku (16 bitů pro každou složku z RGB) a 16 bitový *Alpha* kanál[10]. Stejně jako GIF je vhodný pro webovou grafiku, ale na rozdíl od tohoto formátu nepodporuje možnost animací, naopak podporuje více transparentních barev v obrazu. Pro transparentnost je vymezeno 8 bitů (256 úrovní).



Obr. 24: Výřezy z obrázku ve formátu PNG – 236 kB

TIFF (Tagged Image File Format)

První verze tohoto formátu byla uvedena v roce 1987, naposledy byl upraven v roce 1992 firmou Aldus. Později práva odkoupila firma Adobe. Stejně jako BMP ukládá grafické informace formou bitmapy. Jedná se o velice stabilní formát, tzn. nehrozí ztráta informací při přenosu. Nabízí několik způsobů bezztrátové komprese a ztrátovou kompresi, kterou používá formát JPEG. Dále nabízí možnost použití různé barevné hloubky. Toto všechno dělá z TIFFu velice flexibilní formát, který byl z počátku navržen jako výstup scannerů, dnes je ovšem jeho použití obecnější.



Obr. 25: Výřezy z obrázku ve formátu TIFF – 579 kB

5.1.2 Zjišťování a nastavování DPI

DPI (Dot Per Inch[12] – česky bodů na palec) je vlastnost obrazu, která určuje jeho rozměry na výstupním zařízení. Když má např. obraz o rozměrech 1600 × 1200 nastaveno 300 DPI, pak bude mít obraz po vytisknutí následující rozměry.

$$Výška: \frac{1200}{300} = 4\ palce = 10{,}16\ cm$$
$$Šířka: \frac{1600}{300} \approx 5{,}33\ palců \approx 13{,}55\ cm$$

V hlavičce obrazu, kde je informace o DPI uložena, se může nacházet vertikální a horizontální DPI. Ve většině případů jsou ovšem tyto dvě hodnoty shodné (většina grafických editorů, které umožňují nastavovat DPI také nabízí k nastavení pouze jednu hodnotu, která se pak uloží do obou proměnných).

5.2 Návrh řešení

5.2.1 Grafické formáty

Borland Delphi standardně pracuje s formátem BMP. V modulech, kterými vývojové prostředí disponuje se nachází také modul *jpeg.dcu*, kde je posán i formát JPEG. Ostatní formáty Borland Delphi nepodporuje, ale existují rozšiřující moduly, které tyto formáty popisují a umí je implementovat. Tyto moduly jsou volně ke stažení na internetu. Při práci s jakýmkoliv formátem kromě BMP je stěžejní načíst obraz, přiřadit ho do bitmapy a po zpracování jej opět převést na požadovaný formát a uložit.

5.2.2 Zjišťování a nastavování DPI

Borlad Delphi standardně nepodporují jakoukoliv práci s vlastností DPI obrazu. Tato informace je uložena v hlavičce obrazu a pokud je známo na které pozici se nachází, je možné se pomocí ukazatele dostat na místo v paměti, kde je tato informace zapsána a tam ji podle potřeby zjišťovat a měnit.

5.3 Implementace

5.3.1 Grafické formáty

Načítání a ukládání všech formátu je napsáno ve třídě *TVstupVystup*, která se nachází v modulu *uVstupVystup*. Třída má dvě veřejné metody.

12 Inch – palec. Jednotka míry; 1 palec je roven 2,54 cm.

```
function Otevri(Cesta:TFileName):TBitmap;
function Uloz(Obraz:TBitmap;Cesta:TFileName;Komprese:Byte;
                                              Paleta:Boolean):Boolean;
```

Tyto metody se volají z formuláře a podle typu souboru (identifikace podle přípony) se data předávají soukromým metodám, které provedou požadovanou operaci. Parametr Komprese ve funkci *Uloz* slouží jako předání stupně komprese pro formát JPEG a parametr *Paleta* je určen k rozeznání, zda se má obraz uložit v 8 bitové grafice.

BMP a BMP s 8 bitovou barevnou hloubkou

Načítání a ukládání tohoto formátu je zapsáno v následujících metodách.

```
function OtevriBMP(Cesta:TFileName):TBitmap;
function UlozBMP(Obraz:TBitmap;Cesta:TFileName):Boolean;
function UlozBMP256(Obraz:TBitmap;Cesta:TFileName):Boolean;
```

Ukládání a načítání probíhá standardním způsobem přes členské funkce datového typu *TBitmap* LoadFromFile a SaveToFile. Metoda ukládání v 256 barvách probíhá stejně s tím rozdílem, že se před uložením provede algoritmus na výpočet a aplikaci adaptivní palety, který je popsán v kapitole 2.

JPEG

Implementace je napsána v následujících metodách.

```
function OtevriJPEG(Cesta:TFileName):TBitmap;
function UlozJPEG(Obraz:TBitmap;Cesta:TFileName;
                                            Komprese:Byte):Boolean;
```

Zde je již potřeba načíst zvlášť soubor s obrazem a následně jej přiřadit do datového typu *TBitmap*.

```
var Obraz:TBitmap;
    Jpeg:TJPEGImage;
begin
  Obraz:=TBitmap.Create();
  Jpeg:=TJPEGImage.Create();
  Jpeg.LoadFromFile(Cesta);
  Obraz.Assign(Jpeg);
  Result:=Obraz;
  Jpeg.Destroy();
end;
```

Ukládání se provede opačným způsobem.

```
var JPEG:TJPEGImage;
begin
  Result:=True;
  JPEG:=TJPEGImage.Create();
  JPEG.CompressionQuality:=Komprese;
  try
    JPEG.Assign(Obraz);
    JPEG.SaveToFile(Cesta);
  except
    Result:=False;
  end;
  JPEG.Destroy();
end;
```

GIF[13]

Implementováno v následujících metodách.

```
function OtevriGIF(Cesta:TFileName):TBitmap;
function UlozGIF(Obraz:TBitmap; Cesta:TFileName):Boolean;
```

Zde samotné načítání a ukládání probíhá obdobným způsobem, jako u JPEG s tím rozdílem, že datový typ *TGifImage*, který je stěžejní pro práci s tímto formátem je odvozen od viditelné komponenty *TImage* a při zavolání konstruktoru je třeba jako parametr uvést vlastníka objektu (nějaká jiná komponenta, zpravidla hlavní formulář) a jelikož je funkce zapsána v jiném modulu, je třeba se odkázat přes instanci *Application* (popsáno v modulu *Forms*) do hlavního modulu s formulářem, ze kterého se hlavní formulář uvede jako parametr konstruktoru.

```
GIF:=TGifImage.Create(Application.MainForm);
```

Další práce s tímto formátem je shodná s předchozím postupem u formátu JPEG. Moduly, které umožňují práci s formátem GIF nejsou připraveny na práci s transparentním obrazem, proto načtení takového obrazu neproběhne v pořádku.

13 Moduly ke stažení na http://www.volny.cz/runat/download/download.htm

PNG[14]

Práce s tímto formátem je implementována v následujících metodách.

```
function OtevriPNG(Cesta:TFileName):TBitmap;
function UlozPNG(Obraz:TBitmap;Cesta:TFileName):Boolean;
```

Načítání a ukládání tohoto formátu je naprosto shodné s postupem u formátu JPEG s tím rozdílem, že zde se nenastavuje kompresní stupeň.

TIFF[15]

Implementováno v následujících metodách.

```
function OtevriTIF(Cesta:TFileName):TBitmap;
function UlozTIF(Obraz:TBitmap;Cesta:TFileName):Boolean;
```

Třída *TTiffGraphic* je odvozena od třídy *TBitmap*, tudíž přebírá všechny vlastnosti tohoto standardního datového typu a základní operace jako načítání a ukládání je totožná s klasickým načítáním bitmapy. Chyba nastávala při načítání obrazu ve chvíli, kdy měl obraz barevnou hloubku (v Borland Delphi reprezentována vlastností *PixelFormat*) 32 bitů (pf32Bit). Podle barevné hloubky algoritmus přiřazuje hodnoty atributům *DataOrPointer* a *FBitsPerSample* a zde se počítá s maximální barevnou hloubkou 24 bitů. Po doplnění o možnost 32 bitové barevné hloubky se odstraní chyba a algoritmus načtení obrazu se dokončí správně. Tato změna byla provedena v modulu *GraphicEx*, kde je třída *TTiffGraphic* implementována.

```
case Source.PixelFormat of
      pf1bit:
        begin
          DataOrPointer := 1;
          FBitsPerSample := 1;
        end;
      pf4bit:
        begin
          DataOrPointer := 4;
          FBitsPerSample := 4;
        end;
```

14 Moduly ke stažení na http://files.codes-sources.com/fichier.aspx?id=46569&f=sources%2fpngimage.pas

15 Moduly ke stažení na http://www.koders.com/delphi/fid683F1F4625F55765FD95866D982B69D65D332E78.aspx

```
      pf8bit,
      pf16bit,
      pf32bit,   //dodatecne doplneno
      pf24bit:
        begin
          DataOrPointer := 8;
          FBitsPerSample := 8;
        end;
```

5.3.2 Zjišťování a nastavování DPI

Práce s touto vlastností je zapsána ve třídě TDPI (modul uDPI) a obsahuje tři veřejné metody.

```
function ZjistiDPI(Bitmap:TBitmap):Integer;
procedure NastavDPI(Bitmap:TBitmap; DPI:Integer);
function PalceNaMm(Delka,DPI:Integer):Extended;
```

Jak již bylo výše napsáno, hodnota DPI se dá v Borlad Delphi zjistit a nastavit, pokud je známa pozice tohoto údaje v hlavičce obrazu. U formátu BMP se tato hodnota nachází na pozici 38 (na této pozici uložena informace o horizontálním rozlišení). Protože datový typ *TBitmap* takto procházet nejde, musí se nejprve obraz uložit do datového proudu *TMemoryStream* a pak se dá s obrazem pracovat následujícím způsobem:

```
try
  Result:=0;
  Stream:=TMemoryStream.Create();
  Bitmap.SaveToStream(Stream);
  Stream.Position:=38;
  if Stream.Read(Data,2)=2 then
  begin
    Temp:=Data;
    Result:=Round(Temp/39.370079);
  end;
finally
  Stream.Destroy();
end;
```

Hodnota DPI ukládá do návratové hodnoty (proměnná *Result*). V hlavičce bitmapy je informace o tomto rozlišení uložena jako počet bodů na metr; proto se hodnota musí vy-

dělit číslem39,370079, které vyjadřuje poměr mezi 1 metrem a palcem, neboli jeden metr je roven 39,370079 palcům.

Při nastavování hodnoty DPI je postup obdobný, akorát se opět musí nová hodnota převést na počet bodů na metr a uloží se na pozici ve streamu. Potom se ukazatel posune o 4 bajty[16] a nastaví tutéž hodnotu jako vertikální rozlišení.

```
Stream:=TMemoryStream.Create();
Bitmap.SaveToStream(Stream);
Data:=Round((DPI) * 39.370079);
Stream.Position:=38;
if Stream.Write(Data,2)=2 then
begin
  Stream.Position:=42;
  if Stream.Write(Data,2)=2 then
  begin
    Stream.Position:=0;
    Bitmap.LoadFromStream(Stream);
  end;
end;
```

Tyto postupy jsou převzaty z [3].

Při převodu na metrickou soustavu, aby bylo možné uživateli zobrazit, jak velký bude obraz v milimetrech je důležité vědět, jaké má obraz DPI a počet pixelů (myšleno ve formě výšky, či šířky). Zde je důležité vědět převodní vztah, že 1 palec vyjádřen v milimetrech má hodnotu 2,54. Převodní funkce pak vypadá takto:

```
var Cislo:Extended;
begin
  Cislo:=(Delka/DPI)*(25.4);
  Result:=Cislo;
end;
```

kde je v proměnné Délka je uveden počet pixelů a výsledek se vrátí jako reálné číslo v proměnné *Result*.

5.4 Dosažené výsledky

Rozšíření o načítání a ukládání jiných formátů se zdařilo téměř podle očekávání. Drobné úpravy se musely provést v modulu pro ukládání formátu TIFF. Dále není

16 Datový typ *DWORD* ve kterém jsou zapsány údaje o rozlišení v bodech na metr má délku 4 bajty.

možné načítat transparentní obraz ve formátu GIF a tento formát nelze ukládat (neznámá chyba ve stažených modulech). Nicméně v praktickém využití není jediný důvod, proč v tomto formátu ukládat, tudíž se tento nedostatek nemusí považovat za závažný. Formáty které aplikace bude běžně využívat se dají snadno otevřít i uložit při přijatelné době zpracování (řádově jednotky sekund; v praxi maximálně 2 až 3 sekundy[6]). Více o grafických formátech v [2].

Problematika DPI se povedla vyřešit pouze pro formát BMP. Důvodem je nedostatek informací o hlavičkách ostatních formátů. Práce s tímto údajem trvá zanedbatelnou dobu a lidské vnímání není schopno tento čas ani zaregistrovat.

6 Hlavní aplikace

6.1 Teoretický rozbor

Pro využití v praxi je třeba vytvořit nástroj, který specificky spojí dva obrazy a to tak, že načte dva obrazy a potom bude postupně brát vodorovné pruhy střídavě z obou obrazů a tyto pruhy bude spojovat do výsledného obrazu. Pruh bude mít pro jedno provedení algoritmu stejnou tloušťku, ale pro každé spojení bude volitelné, kolik pixelů má být pruh vysoký.

Výsledek bude sloužit jako reklamní plocha, kde bude možné zobrazit dvě reklamy najednou. Na vytištěný obraz, který byl vytvořen spojením dvou původních obrazů se položí plastová deska s vodorovně zvlněným povrchem. Tento povrch zlomí úhel pohledu na obraz tak, že je vidět pouze jeden obraz. Díky jemnosti pruhů a zvlnění desky se takový obraz lidskému oku jeví jako celistvý, přes to, že skutečně vidí pouze jeho jednu polovinu. Samotné zařízení pro reklamní panel obsahuje ještě elektromotor, který po určitých intervalech s reklamním plakátem pohybuje tak, aby se díky zlomu světla přes vlnitý povrch zobrazila buď první, nebo druhá reklama. Tato technologie má velkou výhodu v tom, že na jedné reklamní ploše poskytuje možnost zobrazit dvě reklamy a to bez použití jakýchkoliv složitých zařízení pro otáčení mnohačlennými segmenty, zde se pouze posunuje papírem řádově v milimetrech. Tato jednoduchá a efektivní technologie reklamních panelů se nazývá Moving Image Display (Obr. 26)[17].

Obr. 26: Moving Image Display panel

17 Zdroj: http://www.tornado.cz/pop-and-pos/moving-image-displej/

6.2 Návrh řešení

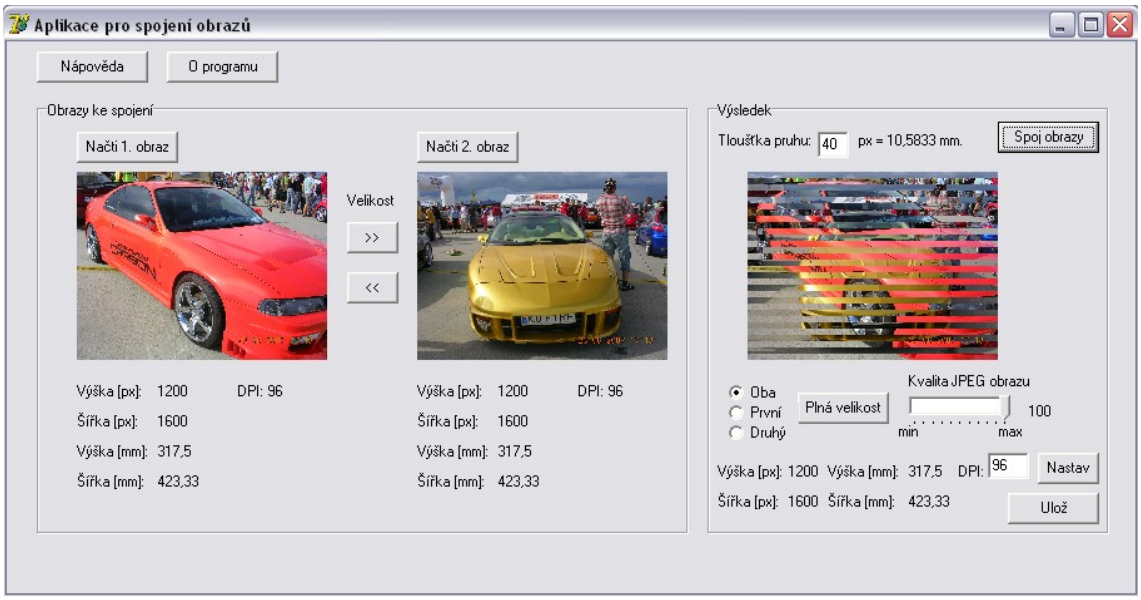
Cílem této části, je vytvořit aplikaci, která dovede vhodně využít poznatky z předchozích kapitol a zároveň nabídne uživateli jednoduché a intuitivní GUI[18] rozhraní, ze kterého se bude aplikace bude ovládat. Měla by uživateli průběžně zobrazovat informace o rozměrech v pixelech i v milimetrech.

Pro ověření, zda se tloušťka pruhu zvolila o vhodné hodnotě bude sloužit funkce, která zobrazí na výsledném obraze pouze část tvořenou jedním obrazem. To dá možnost uživateli jednoduše posoudit, zda budou na takovém obrazu dobře čitelné všechny informace.

Další důležitou věcí je informace o tom, jak velká (v milimetrech) bude tloušťka jednoho pruhu o zvoleném počtu pixelů. Původní návrh byl takový, aby bylo možné tloušťku nastavovat v milimetrech, což je ovšem výpočetně náročná operace (ve většině případů by se musela provést interpolace a úprava DPI), která navíc mění velikost obrazu, tudíž bude vhodnější z počtu pixelů, které reprezentují tloušťku a z DPI obrazu vypočítat, kolik milimetrů vychází na jeden pruh a podle potřeby tloušťku regulovat v pixelech.

6.3 Implementace

Aplikace (Obr. 27) je vytvořena pomocí standardních komponent vývojového prostření Borlad Delphi.



Obr. 27: Náhled aplikace po spojení dvou obrazů

18 GUI (Graphical User Interface) – grafické uživatelské rozhraní.

Načítání je implementováno pomocí třídy *TVstupVystup* popsané v části 5.3.1. Po každém kroku se provede procedura

```
procedure Aktualizuj();
```

která má za úkol aktualizovat všechna číselná data na formuláři (využití třídy TDPI popsané v části 5.3.2) a zároveň kontroluje, zda jsou oba obrazy načteny a zda jsou stejně velké. Pokud ano, povolí se stisknutí tlačítek v rámu výsledného obrazu (spojení, uložení atd.). Tato procedura byla doplněna jako privátní metoda formuláře.

V případě, že se načtou obrazy, které nejsou stejně velké, lze velikost jednoho jednoduše přizpůsobit na velikost druhého, pomocí tlačítek mezi načtenými obrazy. Tyto tlačítka vyvolají interpolaci implementovanou v třídě *TInterpolace*, která je popsaná v kapitole 4.

Po načtení obou obrazů a případném přizpůsobení velikostí se aktivuje tlačítko pro spojení obrazů. Toto spojení využívá metodu CopyRect (část 3.2.3) ze třídy TSpojeni, která je popsaná v kapitole 3.

Pro zobrazení plné velikosti výsledného obrazu, či jen jedné části ze dvou obrazů, je připravený oddělený formulář pro zachování přehlednosti aplikace. Pro zobrazení jednoho ze dvou spojených obrazů je určena třída *TPrepinac*, která využívá také kopírování pomocí *CopyRect* (část 3.2.3) s tím rozdílem, že spojuje pouze jeden obraz s prázdnou bitmapou nastavenou na velikost obrazu. Aplikace dává možnost nastavení DPI (třída *TDPI* popsaná v části 5.3.2). Po dostatečných úpravách je možno obraz uložit v několika formátech. Zde se opět využívá třída *TVstupVystup* popsaná v části 5.3.1. Jednou z možností uložení je i formát BMP s barevnou hloubkou 8 bitů (256 barev). Při tomto uložení se provede algoritmus pro vytvoření adaptivní palety, na který je zaměřena kapitola 2.

6.4 Finanční přínos

Bez této aplikace byla firma nucena tisknout reklamní plakáty u partnerské firmy v Číně, která zajistila spojení obrazů. V následující tabulce (Tabulka 4) je srovnání situace, kdy měl zákazník požadavek na takovou reklamu před tím, než měla firma k dispozici aplikaci pro spojení obrazů a situace po té. Údaje uvedené v tabulce jsou pouze orientační.

	Cena tisku	*Cena za spojení*	*Cena za dopravu*	*Čas dodání*	*Dohled na kvalitu*
Dříve (Čína)	300 Kč	300 Kč	5000 Kč[19]	min. 15 dní	minimální
Nyní (ČR)	500 Kč	0 Kč	50 Kč	ihned	maximální

Tabulka 4

Zdroj: tornado a. s.

6.5 UML diagram

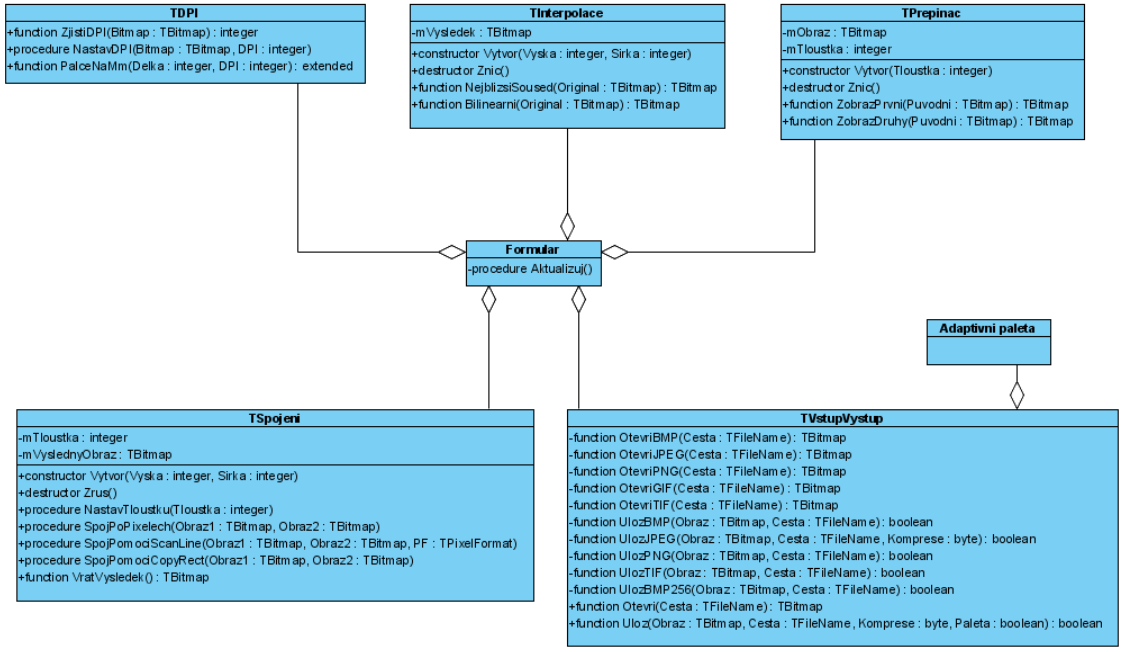


Diagram je ve větší podobě v programátorské dokumentaci, kde jsou i popisky k jednotlivým atributům a metodám.

6.6 Dosažené výsledky

Většina požadavků na aplikaci byla splněna, práce je rychlá a ovládání jednoduché. Všechny moduly se povedlo bez problému implementovat. Největší nedostatek aplikace je nastavení a zjištění DPI pouze pro formát BMP. Nicméně pokud uživatel nastaví nějakou hodnotu DPI i přes to, že předem ví, že obraz nebude ukládat do formátu BMP, poslouží mu přepočty jako kalkulačka, pomocí které může poměrně přesně nastavit požadovanou tloušťku pruhu.

19 Tato cena je relativní; pokud bude objednávka na 100 ks, bude cena dopravy rozdělena do celého počtu, ovšem zákazník s požadavkem na jeden plakát není výjimkou.

7 Závěr

Tato práce se zabývala vytvořením aplikace pro specifické spojení obrazů. Důvodem byla potřeba takové aplikace v praxi, kde není běžně dostupná. Kromě tohoto stěžejního úkolu si práce kladla ca cíl implementaci vybraných algoritmů z oblasti počítačové grafiky. Implementace byla nutná vzhledem k tomu, že grafická podpora v knihovně VCL tyto algoritmy nezahrnuje.

Většina původních cílů byla splněna; aplikace splňuje téměř všechny požadavky, které byly před samotnou implementací vzneseny a je napsána s využitím technik objektově orientovaného programování, což dává možnost znovupoužitelnosti některých tříd.

Povedlo se implementovat algoritmus na výpočet adaptivní palety postupem, jaký je uveden v [1]. Zde byl ovšem postup pouze naznačen. Některé dílčí algoritmy (např. jak vhodně zmenšovat 3D oblasti) ve zdroji nebyly popsány a bylo třeba navrhnout vhodné metody, které by požadované operace provedly co nejefektivněji. Tento úkol se v závěru projevil jako nejsložitější část práce. Stejně tak není ve zdroji popsáno, jakým způsobem paletu přiřadit obrazu, pro který byla vypočítána. To znamenalo detailní nastudování některých vlastností třídy *TBitmap* o kterých se běžně dostupná literatura nezmiňuje. Všechny tyto dílčí úkoly se podařilo vyřešit a celkový výsledek se dá považovat za uspokojivý.

Hlavní činnost aplikace, tedy spojení obrazů, je specifická operace, pro kterou bylo třeba vymyslet nějaký efektivní algoritmus. Standardní práce s pixely, tedy přes indexy, se ukázala jako značně neefektivní z důvodu rychlosti zpracování (řádově desítky sekund), ovšem podařilo se implementovat alternativní řešení založené na kopírování datových bloků, které tento problém odstranilo a doba zpracování se zkrátila na desetiny sekund, což se dá považovat za uspokojivý výsledek.

Z vybraných metod interpolace se podařilo implementovat dvě nejznámější; interpolaci nejbližším sousedem a bilineární interpolaci. Interpolace nejbližším sousedem je sice rychlá a jednoduchá, ovšem pro potřeby aplikace je vhodnější bilineární. Její zpracování ovšem trvá dlouho (řádově desítky vteřin). Zde by se dalo v práci pokračovat hledáním lepšího způsobu zpracování, který interpolaci provede rychleji. Další možností, jak v této kapitole pokračovat je implementace bikubické interpolace.

Rozšířit aplikaci o načítání a ukládání obrazů ve vybraných standardních formátech se zdařilo prostřednictvím modulů nalezených na internetu. V případě formátu TIFF se musel provést drobný zásah do původního kódu modulu, aby bylo možné načítat tento formát i případě, že má obraz barevnou hloubku 32 bitů.

Zjišťování a nastavování DPI se zdařilo pouze u formátu BMP. U ostatních formátů se toto nezdařilo kvůli nedostatku informací o hlavičkách obrazových souborů, ze kterých jde vypočítat na které pozici se tato informace nachází.

Hlavní aplikace má podobu jednoduše ovladatelného formuláře, který využívá standardní GUI[18] komponenty. Aplikace disponuje nápovědou, která se věnuje všem důležitým prvkům na formuláři, takže by měla s jejím využitím být snadno ovladatelná i pro začátečníka v práci s počítačem. Obsahuje a využívá všechny moduly, které jsou v práci popsány.

8 Zdroje

[1] ŽÁRA, J. BENEŠ, B, SOCHOR, et al. *Moderní počítačová grafika* 2. přeprac. vyd. Brno : Computer Press, 2004. 609 s. ISBN 80-251-0454-0.
dostupné: univerzitní knihovna Univerzity Pardubice

[2] MURRAY, J, RYPER, W, *Encyklopedie grafických formátů*. Praha : Computer Press, 1995. 736 s. ISBN 80-85896-18-4.
dostupné: univerzitní knihovna Univerzity Pardubice

[3] SVOBODA, L, VONEŠ, P, KONŠAL, T, et al. *1001 tipů a triků pro Delphi*. Praha : Computer Press, 2001. 390 s. ISBN 80-7226-529-6.
dostupné: univerzitní knihovna Univerzity Pardubice

[4] SKŘIVAN, J. *GIF, JPEG a PNG - jak a kdy je použít?* [online]. 2002 , 16. 5. 2002 [cit. 2008-03-18]. Dostupný z WWW: <http://interval.cz/clanky/gif-jpeg-a-png-jak-a-kdy-je-pouzit/>. ISSN 1212-8651.

[5] *The Incredible Programs - Download page* [online]. 2008 [cit. 2008-05-02]. Dostupný z WWW: <http://www.volny.cz/runat/download/download.htm>.

[6] *Sources/pngimage.pas* [online]. 2008 [cit. 2008-05-02]. Dostupný z WWW: <http://files.codes-sources.com/fichier.aspx?id=46569&f=sources/pngimage.pas>.

[7] *Koders Code Search: GraphicEx.pas - Delphi* [online]. 2008 [cit. 2008-05-02]. Dostupný z WWW: <http://www.koders.com/delphi/fid683F1F4625F55765FD95866D982B69D65D332E78.aspx>.