Grafický editor pro zpracování bitmapových souborů

Bc. Michael Borkovec

Diplomová práce
2006

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav aplikované informatiky
akademický rok: 2005/2006

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Michael BORKOVEC**
Studijní program:	**N 3902 Inženýrská informatika**
Studijní obor:	**Informační technologie**
Téma práce:	**Grafický editor pro zpracování bitmapových souborů**

Zásady pro vypracování:

1. Vypracujte literární rešerši na zadané téma. Ta bude obsahovat charakteristiku jazyka Java a základní informace o nejrozšířenějších bitmapových souborech.
2. Navrhněte a naprogramujte aplikaci v jazyce Java pro zobrazování a úpravy bitmapových souborů.
3. Prostředí programu bude navrženo tak, aby umožňovalo vytváření a úpravy 2D-grafických obrazů, které půjde ukládat do bitmapových souborů.
4. S využitím tříd z knihovny java.awt.image bude tato aplikace podporovat konverzi základních bitmapových formátů.
5. Naprogramujte algoritmy, které budou na obrazy aplikovat vybrané efekty a filtry.

Rozsah práce:

Rozsah příloh:

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

Seznam odborné literatury:

[1] ŽÁRA, J., BENEŠ, B., SOCHOR, J., FELKEL, P.: Moderní počítačová grafika, Computerpress, Brno, 2004. ISBN 80-251-0454-0
[2] MARTIŠEK, D.: Matematické principy grafických systémů, Littera, Brno, 2002. ISBN 80-85763-19-2
[3] HEROUT, P.: Java - bohatství knihoven, Kopp, České Budějovice, 2003. ISBN 80-7232-209-5
[4] HAWLITZEK, F.: Java2 - příručka programátora, Grada, Praha, 2002. ISBN 80-247-9060-2
[5] KISZKA, B.: 1001 tipů a triků pro programování v jazyce Java, Computerpress, Brno, 2003. ISBN-80-7226-989-5

Vedoucí diplomové práce: **Ing. Pavel Pokorný, Ph.D.**
Ústav aplikované informatiky

Datum zadání diplomové práce: **14. února 2006**

Termín odevzdání diplomové práce: **26. května 2006**

Ve Zlíně dne 14. února 2006

prof. Ing. Vladimír Vašek, CSc.
pověřený děkan

doc. Ing. Ivan Zelinka, Ph.D.
ředitel ústavu

ABSTRAKT

Tato diplomová práce popisuje jak vytvářet počítačovou rastrovou grafiku pomocí programovacího jazyka Java. V první části jsou probrány základní vlastnosti počítačové grafiky. Poté jsou zmíněny nejčastěji používané obrazové filtry a následující část vytváří s touto problematikou spojitost v programovacím jazyce Java. Praktická část popisuje obrazový editor *Imagine*, který byl speciálně programován pro tuto diplomovou práci. Tento program je schopen pracovat s obrazovými soubory, vytvářet jednoduchou grafiku a aplikovat vlastní filtrační algoritmy.

Klíčová slova: *Java, JPEG, BMP, PNG, RGB, konvoluce, matice, lineární, nelineární, filtr, histogram, medián.*

ABSTRACT

Purpose of this thesis is to describe how to create computer raster graphics in Java programming language. There are clarified basic computer graphic characteristics in the first part of this text. The most often used image filters are mentioned. Following part makes a connection with this subject in Java language. Practical section describes image editor *Imagine*, which was especially programmed for this thesis. This program is able to operate with image files, create simple graphics and apply its own filtering algorithms.

Keywords: *Java, JPEG, BMP, PNG, RGB, convolution, matrix, filter, linear, nonlinear, filtr, histogram, median.*

Na tomto místě bych chtěl poděkovat vedoucímu diplomové práce - Ing. Pavlu Pokornému, Ph.D., mé rodině, mým přátelům a také všem, kteří měli dobré připomínky k mému programu a textu.

„Vzdělání má hořké kořeny, ale sladké ovoce.“

Démokritos z Abdér

OBSAH

ÚVOD **9**

I TEORETICKÁ ČÁST **10**

1 ÚVOD DO POČÍTAČOVÉ GRAFIKY **11**

1.1 BAREVNÉ MODELY 11

1.1.1 Binární 11
1.1.2 16 Barev 11
1.1.3 256 barev 11
1.1.4 Indexový mód 12
1.1.5 Odstíny šedi 12
1.1.6 RGB(A) 12
1.1.7 Barevný model CMY(K) 14
1.1.8 Ostatní barevné modely 14

1.2 ROZDĚLENÍ PODLE DRUHU OBRAZOVÝCH DAT 15

1.2.1 Rastrový formát 15
1.2.2 Vektorový formát 16

1.3 POPIS RASTROVÝCH SOUBORŮ [10] 17

1.3.1 BMP 17
1.3.2 JPEG 18
1.3.3 PNG 18

2 ČASTO POUŽÍVANÉ ALGORITMY V POČÍTAČOVÉ GRAFICE **20**

2.1 TRANSFORMACE BAREV 20

2.1.1 Omezení barevného prostoru 20
2.1.2 Barevná paleta 21

2.2 GEOMETRICKÉ TRANSFORMACE DISKRÉTNÍHO OBRAZU 21

2.2.1 Lineární geometrické transformace obrazu [13] 22
2.2.2 Warping a morfing 24

2.3 ALGORITMY PRO LOKÁLNÍ ÚPRAVY V OBRAZE 24

3 ALGORITMY PRO ÚPRAVU OBRAZU VYUŽÍVAJÍCÍ HISTOGRAM **26**

3.1 EKVALIZACE HISTOGRAMU 27

3.2 KONTRAST A JAS 29

3.3 NEGATIV 30

3.4 PRAHOVÁNÍ 30

3.5 GAMA KOREKCE 31

4 FILTRACE OBRAZŮ **32**

4.1 LINEÁRNÍ FILTRY 32

4.1.1 Dolní propust 33
4.1.2 Horní propust 34

4.2 NELINEÁRNÍ FILTRY 34

4.2.1 Filtr minimum 34

4.2.2 Filtr medián 35
4.2.3 Filtr maximum 35
5 CHARAKTERISTIKA PROGRAMOVACÍHO JAZYKA JAVA 36
5.1 POROVNÁNÍ S OSTATNÍMI JAZYKY 36
5.1.1 Vlastnosti Javy 36
5.1.2 Porovnání výkonu s programy v C# 38
5.2 PROGRAMÁTORSKÁ PODPORA JAVY 38
5.2.1 Kategorie Javy 39
5.2.2 Zdroje informací o Javě 39
5.3 VÝVOJOVÁ PROSTŘEDÍ PRO JAVU 40
5.3.1 NetBeans 40
5.3.2 Eclipse 41
5.3.3 BlueJ 41
5.3.4 JBuilder 42
5.4 TVORBA GRAFICKÉHO UŽIVATELSKÉHO ROZHRANÍ POMOCÍ SWING 42
5.5 APLETY 43
5.6 KNIHOVNA JAVA.AWT.IMAGE [15] 43
5.6.1 Třída BufferedImage 43
5.6.2 Třída ColorModel 44
5.6.3 Třída ConvolveOp 44
5.6.4 Třída Kernel 44
5.6.5 Třída PixelGrabber 45
5.6.6 Rozhraní ImageConsumer 45
5.6.7 Rozhraní ImageProducer 45
5.7 TŘÍDA GRAPHICS2D [15] 45
II PRAKTICKÁ ČÁST 47
6 GRAFICKÝ EDITOR IMAGINE 1.0 48
6.1 SPUŠTĚNÍ A OVLÁDÁNÍ PROGRAMU 48
6.1.1 Instalace a spuštění programu 49
6.1.2 Hlavní okno programu 49
6.1.3 Otevření souboru 49
6.1.4 Kreslení jednoduchých tvarů 49
6.1.5 Algoritmy pro úpravu obrazu 50
6.2 PRÁCE S ALGORITMY 50
6.2.1 Lineární filtry 50
6.2.2 Nelineární filtry 51
6.2.3 Expozice obrazu 52
6.2.4 Negativ obrazu 53
6.2.5 Ekvalizace obrazu 53
6.3 ORIGINÁLNÍ „JAVA“ FILTRY 53
6.4 ARCHITEKTURA PROGRAMU IMAGINE 1.0 54
6.4.1 Notace programového kódu 56

6.5 IMAGINE.JAVA 57
6.6 PROPERTIESINTFRAME.JAVA 58
6.7 INOUTMANAGER 58
6.8 IMAGEFRAME – HLAVNÍ ČÁST PROGRAMU 58
6.8.1 Funkce paintCompoment() 58
6.8.2 Události od myši 59
7 VLASTNÍ ALGORITMY 61
7.1 CONVOLVEFILTERS – LINEÁRNÍ FILTRY 61
7.2 MYHISTIMAGE – EXPOZICE OBRAZU A NELINEÁRNÍ FILTRY 61
7.2.1 Funkce passImg() 61
7.2.2 Funkce exposureImg() 62
7.2.3 Funkce NonLinearFilters() – nelineární filtry 66
7.3 UKÁZKOVÉ APLIKACE VLASTNÍCH ALGORITMŮ NA OBRAZ 67
7.3.1 Lineární filtry 68
7.3.2 Nelineární filtry 69
7.3.3 Expozice obrazu – křivky 71
7.3.4 Negace 72
7.3.5 Ekvalizace 74
ZÁVĚR 75
SEZNAM POUŽITÉ LITERATURY 77
SEZNAM OBRÁZKŮ 79

ÚVOD

Současná počítačová grafika zahrnuje především tyto oblasti:

- algoritmy pro kresbu, zobrazení a uchování 2D a 3D vektorových obrázků, včetně animovaných sekvencí.
- tvorbu grafických materiálů prostřednictvím počítačové techniky.
- uchování a zpracování bitmapové, čili rastrové grafické informace.
- zpracování obrazů – pro 2D či 3D rastrovou grafickou informaci.
- metody pro uchování a zpracování rastrových dat, metody pro extrakci žádoucí informace z obrázku.

Kvůli své výpočetní náročností je to právě počítačová grafika, která je hnací silou v rozvoji informačních technologií. Z jedné strany jsou to zvyšující se požadavky v oblastech prezentace a marketingu – webové stránky a programy. Z druhé strany je to rostoucí poptávka po relaxaci – počítačové hry a video. Na tyto vysoké nároky reagují výrobci hardware zvyšováním výkonu počítačů a softwarové firmy zdokonalováním grafických algoritmů. Díky tomuto vývoji je dnes možné využívat i mobilní telefony k fotografování, k trojrozměrným barevným hrám, či ke sledování videa.

Díky své univerzálnosti lze programovací jazyk Java používat v aplikacích pro mobilní telefony, v běžných uživatelských programech pro PC na všech operačních systémech, i v náročných serverových aplikacích. Z těchto důvodů je patrné, že v jazyce Java se ukrývají velké možnosti využití. Tato práce má za úkol alespoň část těchto možností ukázat podrobněji.

I. TEORETICKÁ ČÁST

1 ÚVOD DO POČÍTAČOVÉ GRAFIKY

Dvourozměrná (2D) počítačová grafika se dá rozdělit podle dvou aspektů. Prvním je reprezentace obrazu, buď pomocí rastru, nebo podle vektorového vyjádření. Druhým aspektem je pak barevný model, který je použit.

1.1 Barevné modely

Přirozeným barevným modelem, je ten, podle kterého vnímá lidské oko.

Lidské oko vnímá [9]:

- Dominantní vlnovou délku - Odstín (*Hue*)
- Čistotu barvy – Sytost (*Saturation*)
- Intenzitu – Jas (*Brightness*)

1.1.1 Binární

Pokud je každý pixel na obrazovce popsán jediným bitem, pak je takový obraz *monochromatický* neboli černobílý. Toto značení má své historické důvody a používá se, i když je poněkud zavádějící, protože binární informace nemusí reprezentovat právě bílou a černou barvu, ale dvě libovolné barvy.

1.1.2 16 Barev

Barva každého bodu je uložena ve čtyřech bitech, tj. v jednom bajtu je uložena barva dvou bodů. Standardní paleta obsahuje těchto šestnáct barev (viz. Obrázek 1).

Obr. 1: Barevný rozsah 16-ti barev

1.1.3 256 barev

Barva každého bodu je uložena v jednom bajtu. Existuje tedy 256 současně zobrazitelných barev. Tyto barvy však mohou být vhodně vybírány z 262 144 =64*64*64 možností pomocí palety.

1.1.4 Indexový mód

Indexový mód je spojen s používáním tzv. *barevné palety*, neboli *mapy barev*. U takového obrazu neprezentuje hodnota pixelu barvu, ale odkazuje do tabulky barev - barevné palety. ***Barevná paleta*** je převodní tabulka, která označuje konkrétní barvu pixelu s daným indexem.

Například u palety *3-3-2* je index reprezentován jedním bytem, a proto má paleta maximálně 256 řádků, jednotlivé barvy mají rozděleny počet úrovní. U červené a zelené 3 bity a u modré jen 2 bity. Výsledný počet barev je 8*8*4=256 barev. [1]

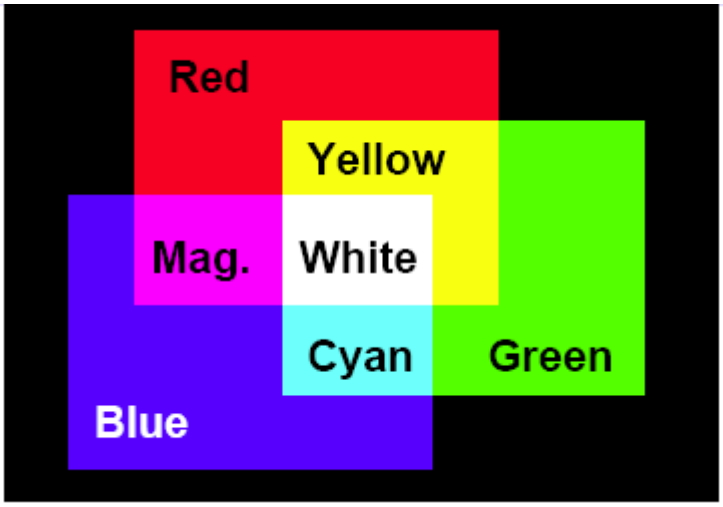
Paleta typu *8-8-8* reprezentuje barvu ve třech bytech, po jednom pro každý barevná kanál. Tím je určeno, že každý bod obrazu může nabývat některé z 256 barev, které se vybírají z celkového množství 2^{24} barev. Tomuto způsobu se říká *pseudo-color.*

1.1.5 Odstíny šedi

Každý bod v obraze reprezentuje buď přímo odstín šedi – *staticgrey*, nebo je opět odkazem do palety – *gray scale*.

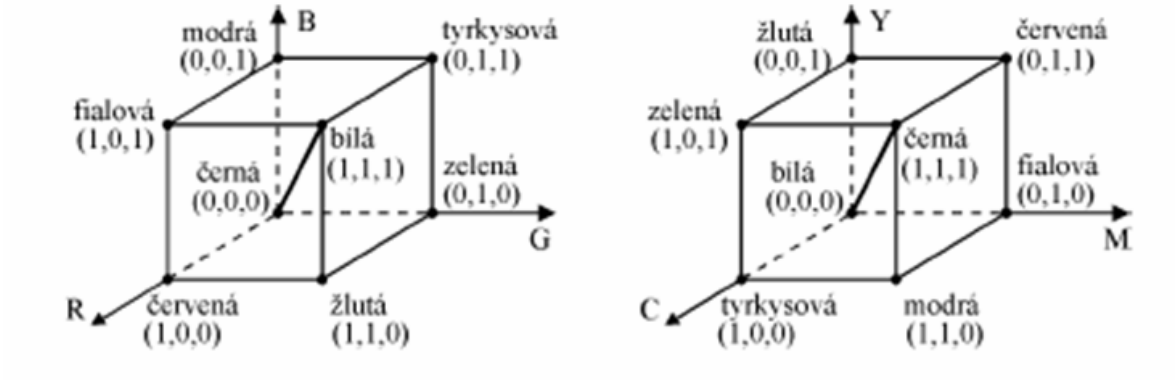
1.1.6 RGB(A)

Barevný model RGB neboli červená-zelená-modrá je ***aditivní způsob*** míchání barev používaný ve všech monitorech a projektorech (jde o míchání vyzařovaného světla).



Obr. 2: Aditivní způsob skládání barev u RGB

Každá barva je udána mohutností tří základních barev - komponent (***červené - red, zelené - green a modré - blue -*** odtud RGB). Model RGB je možné zobrazit jako krychli, kde každá z kolmých hran krychle udává škálu mohutností barevných složek tak, jak ukazuje obrázek č. 3. Potom libovolný bod se souřadnicemi (r,g,b) v této krychli udává hodnotu výsledné barvy. Čím větší je součet mohutností, tím světlejší je výsledná barva.



Obr. 3: Rozdíly ve vyjádření modelů RGB(vlevo) a CMY(vpravo)

Barvy lze vyjádřit trojicí, barevným vektorem, který nabývá hodnot z intervalu <0; 1>. Používá se i značení, kde každá složka nabývá hodnot v intervalu <0; 255>. Každá složka je reprezentována jedním bytem a může tedy nabývat 256 hodnot (2^8 stavů). Celkové množství barev u toho modelu pak je 2^{24}. Čili 16 777 216. Kromě tohoto rozsahu se používají modely s větším i menším rozsahem (např. 12 nebo i 16 bitů na kanál).

True color obraz obsahuje přímo barevné hodnoty v jednotlivých pixelech. Naproti tomu barevný mód označovaný jako ***direct color*** pracuje s RGB hodnotami, které neslouží k zobrazení, nýbrž jsou odkazem do barevných palet pro každou jednotlivou barevnou složku. Při zobrazení musí tedy být k dispozici tři palety, pro každý barevný kanál jedna. Tento způsob je výhodný zejména proto, že umožňuje snadnou změnu všech barev, aniž by se museli měnit hodnoty pixelů v obrazu a využívá se například při gama korekci.

Barevný model ***RGBA*** používá aditivní způsob míchání barev a vychází z modelu RGB, který je rozšířen o alfa kanál A s informací o průhlednosti konkrétního pixelu. Alfa kanál má většinou rozlišení 8 bitů a dokáže rozlišit 2^8=256 úrovní průhlednosti.

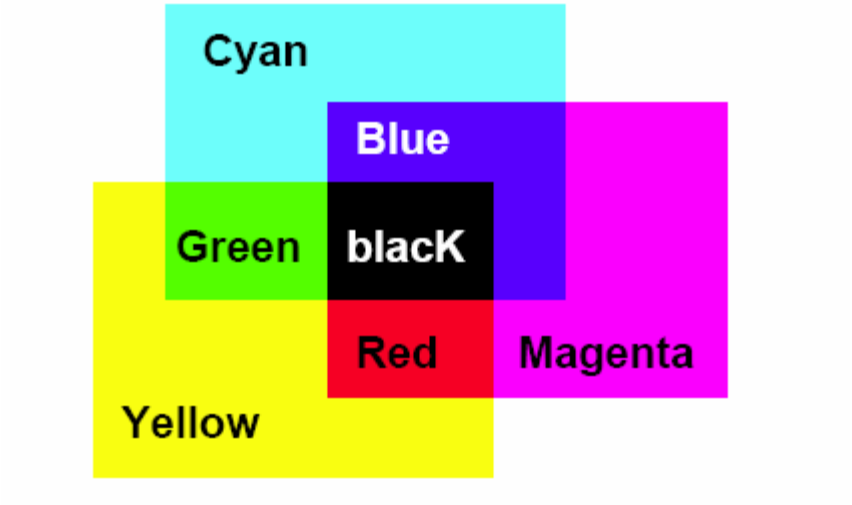
Lidské oko vnímá různým způsobem intenzitu jednotlivých barevných složek (nejcitlivější je na zelenožlutou), takže se používá pro výpočet jasu empirický vztah [1]:

$$I = 0{,}299 \cdot R + 0.587 \cdot G + 0.114 \cdot B$$

1.1.7 Barevný model CMY(K)

CMYK je barevný model založený na ***subtraktivním míchání*** barev (mícháním od sebe se barvy odčítají, tedy omezuje se barevné spektrum, které se odráží od povrchu). CMYK se používá především u reprodukčních zařízení, která barvy tvoří mícháním pigmentů (např. inkoustová tiskárna). Model obsahuje čtyři základní barvy:

azurovou - *Cyan*, purpurovou – *Magenta*, žlutou – *Yellow*, černou - *blacK*, označovanou také jako klíčovou (*Key*).



Obr. 4: Subtraktivní způsob

skládání barev u modelu CMYK

V ideálním případě by byly postačující pouze první tři barvy - model **CMY**, jejichž subtraktivním složením dohromady by měla vzniknout černá barva. Ve skutečnosti však při použití běžných barviv vzniká barva tmavě šedivá, a zároveň je narozdíl od ostatních barev černá výrazně levnější, proto většina tiskových zařízení používá ještě čtvrtou - černou barvu.

1.1.8 Ostatní barevné modely

Modely RGB a CMY jsou přímo použitelné pro odpovídající technická zařízení, ale nejsou až tak blízké intuitivnímu popisu barev. První z modelů, které jsou blízké k lidskému chápání světla, se nazývá ***HSV***. Jeho základními proměnnými jsou **barevný tón - *Hue*, sytost - *Saturation* a jasová hodnota - *Value***. Barevný tón označuje převládající spektrální barvu, sytost určuje příměs jiných barev a jas určuje množství bílého (bezbarvého) světla. Pro prostorové zobrazení tohoto modelu se nepoužívá krychle, ale

šestiboký jehlan, jehož vrchol je umístěn do počátku souřadnicového systému a podstava je otočena kolmo vzhůru.

O něco vhodnějším barevným modelem je pak model ***HLS***, jehož základními proměnnými jsou **barevný tón - *Hue*, světlost - *Lightness* a sytost - *Saturation*.**

Dalšími barevným modelem je L*a*b, který je oblíbený hlavně mezi profesionálními grafiky. Barevné modely YUV, YIQ a YC_BC_R se naproti tomu používají velice často, protože jsou používány v televizním přenosu.

Barevný model YC_BC_R se používá pro přenos TV signálu v normě SECAM a také se používá při ukládání obrazových dat u formátu JPEG, přepočet mezi tímto prostorem a prostorem RGB je [1]:

$$\begin{bmatrix} Y \\ C_B \\ C_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

1.2 Rozdělení podle druhu obrazových dat

Jednodušší možností, jak uchovávat obrazová data je využití rastru, neboli ***matice***[1]. Poněkud složitější je pak reprezentace vektorová.

1.2.1 Rastrový formát

Data jsou reprezentována dvourozměrnou maticí bodů (pixelů), z nichž každý nabývá hodnot podle typu obrazu.

Výhody - Jednoduchá změna ve velikosti souboru. Možnost zpracování jednotlivých pixelů.

Nevýhody - Problémy při aplikaci geometrických transformací. Nevhodné pro plotr.

Rastrové soubory - BMP, JPG, JP2, GIF, PNG, PCX, TGA, WBMP, TIFF …

---

[1] V anglické literatuře „matrix“, s významy jako živná půda, či lisovnice. V tomto případě matrix znamená číselná matice. Pokud máte na mysli film, tak „matrix“ znamená základní hmota.

Programy používající rastrový formát – ACDSee, Adobe Photoshop, Corel Photopaint, Gimp ....

1.2.2 Vektorový formát

Obrazová informace je uložena pomocí vektorů a matematických funkcí.

Výhody - pokud v obraze převládají jednoduché tvary, tak je paměťově méně náročnější než rastrový formát obrazu. Využívá se například při technických výkresech v CAD aplikacích. Při změně měřítka, nebo při rotaci nedochází ke zkreslení. Viz obr. č.5. Podstatnou výhodou vektorového formátu pak je, že uživatel může upravovat tvary v jednotlivých objektech v obraze.

Nevýhody – je nevhodný pro reprezentaci chaotických dat. Například pro fotografii krajiny (rostliny obsahují málo geometrických tvarů). Při ***vektorizaci*** - proces převodu do vektorového formátu[2] pak vzniká příliš velké množství objektů a reprezentace takovýchto dat je pak extrémně náročná na paměť. Například při vektorizaci fotografie města, která měla rozlišení 2592*1944 pixelů vznikl vektorový soubor o velikosti 88 MB a následná práce v programu CorelDraw si vyžádala dalších asi 500MB v operační paměti.

Z těchto důvodů se vektorové formáty s úspěchem používají při vytváření log, nápisů, různých znaků, etiket, ale také při definici fontu písma.

Vektorové soubory - DWG, EPS, CDR, AI, …

Programy pro editaci vektorových souborů - CorelDraw (*.CDR), Adobe Illustrator (*.AI), Zoner Callisto.

[2] Jedním z populárních programů je CorelTrace.

Obr. 5: Zvětšení - vlevo u rastrového

a vpravo u vektorového obrázku

1.3 Popis rastrových souborů [10]

1.3.1 BMP

Formát *BMP* byl poprvé představen v roce 1988 jako součást nového systému OS/2 verze 1.10 SE. O něco později firma Microsoft trochu rozšířila jeho definici a zahrnula ho do svého tehdy nejprodávanějšího 16-bitového grafického operačního prostředí – Microsoft Windows 3.0. Na počátku roku 1992 firma IBM uvedla na trhu první 32-bitový systém OS/2 verze 2.0, který obsahoval vylepšenou variantu BMP s novou strukturou pro uskladnění vícenásobných bitových map v jednom souboru. Tento typ souboru se často obecně označuje jako bitmapové pole.

Obrázky BMP jsou ukládány po jednotlivých pixelech, podle toho, kolik bitů je použito pro reprezentaci každého pixelu je možno rozlišit různé množství barev: 2 (1 bit), 16 (4 bity), 256 (8 bitů), 65 536 (16 bitů), nebo 16,7 miliónu (24 bitů). Osmibitové obrázky mohou místo barev používat šedou škálu.

Soubory ve formátu BMP většinou nepoužívají žádnou kompresi (přestože existují i varianty používající kompresi RLE). Z tohoto důvodu jsou obvykle BMP soubory mnohem větší než obrázky stejného rozměru, které kompresi používají.

Velikost nekomprimovaného obrázku v bytech lze přibližně vypočítat podle vzorce.

(šířka v pixelech) * (výška v pixelech) * (bitů na pixel / 8)

K velikosti obrázku je třeba ještě připočítat velikost hlavičky souboru, která se liší dle jeho verze i dle použité barevné hloubky.

1.3.2 JPEG

JPEG je standardní metoda ztrátové komprese používané pro ukládání počítačových obrázku ve fotorealistické kvalitě. Formát souboru, který tuto kompresi používá, se také běžně nazývá JPEG. Nejrozšířenější příponou tohoto formátu je .jpg, .jpeg, .jfif, .jpe, nebo tato jména psána velkými písmeny.

Skutečným názvem typu souboru je ***JFIF***, což znamená *JPEG File Interchange Format*. Zkratka *JPEG* znamená *Joint Photographic Experts Group*, což je vlastně konsorcium, které tuto kompresi navrhlo.

Když se běžně hovoří o souboru JPEG, míní se tím většinou soubor JFIF, nebo soubor Exif JPEG. Existuje však více formátů soborů založených na kompresi JPEG, například JNG.

JPEG/JFIF je nejčastější formát používaný pro přenášení a ukládání fotografií na internetu. Není však vhodný pro perokresbu, zobrazení textu nebo ikonky, protože kompresní metoda JPEG vytváří v takovém obrazu viditelné a rušivé artefakty. Pro takové účely se většinou používají soubory PNG a GIF. Protože má GIF pouze 8 bitů na pixel, není vhodný pro barevné fotografie, PNG je možné použít pro ukládání fotografií, ale výsledná velikost souboru je nevhodná pro publikování na webu.

Komise JPEG vytvořila vlastní standard na bázi waveletů zvaný *JPEG2000*, od kterého se očekává, že nakonec nahradí originální JPEG standard. Novější ztrátové kompresní metody, zvláště *waveletová* komprese, dávají lepší výsledky. Nicméně JPEG je velmi dobře zavedený standard, který je schopno otvírat mnoho aplikací. Mnoho waveletových algoritmů je navíc patentováno, takže je obtížné, nebo nemožné, používat je v mnoha softwarových projektech.

1.3.3 PNG

PNG (*Portable Network Graphics*) - anglicky přenosná síťová grafika; oficiální výslovnost zkratky je „ping“) je grafický formát určený pro bezeztrátovou kompresi rastrové grafiky. Byl vyvinut jako zdokonalení a náhrada formátu GIF, který je patentově chráněný. PNG nabízí podporu 24 bitové barevné hloubky, nemá tedy jako GIF omezení na maximální počet 256 barev současně. PNG tedy do jisté míry nahrazuje GIF, nabízí více barev a lepší kompresi. Navíc obsahuje osmibitovou průhlednost (tzv. alfa kanál), to

znamená, že obrázek může být v různých částech různě průhledný. PNG však neumí jednoduché animace, které naopak umožňuje formát GIF. PNG se stejně jako formáty GIF a JPEG používá na Internetu.

2 ČASTO POUŽÍVANÉ ALGORITMY V POČÍTAČOVÉ GRAFICE

Tato kapitola se věnuje nejčastěji používaným úpravám obrazu. V následující kapitole (3) jsou podrobněji popsány algoritmy pro úpravu histogramu, kontrastu, jasu, vytvoření negativu, prahování a gama korekci. V další kapitole (4) jsou pak popsány lineární a nelineární filtrace obrazu.

2.1 Transformace barev

Tento požadavek nevychází pouze z potřeby "vylepšování" obrazu, ale je nutný například při přípravě obrazu před tiskem. Tehdy je obraz upravován s ohledem na technologii barevného tisku a často se snižuje celkový počet použitých barev.

2.1.1 Omezení barevného prostoru

Často používané barevné rozlišení ***truecolor*** představuje více než 16 miliónů různých barevných odstínů Velice často pak je potřeba toto množství barev snížit. Například kvůli snížení velikosti souboru, nebo kvůli tisku. Při snižování počtu barev dochází ke ztrátě informace. **Polotónování -** ***halftoning*** je proces, který se používá hlavně u tiskáren a umožňuje snížit počet barev tak, že jeden barevný pixel původního obrazu je převeden na matici bodů s výrazně menším počtem barev [1]. Při halftoningu tedy dochází ke zvětšení obrazu (tzn., že ke zmenšení velikosti souboru je *halftoning* nevhodný). **Rozptylování -** ***dithering*** je vhodný k zobrazení obrazu, který byl například vypočítán v kvalitě 24 bitů na pixel a který má být vykreslen na obrazovku v nezměněné velikosti. Halftoning se dá chápat jako konkrétní případ ditheringu. Tyto metody využívají schopnosti lidského oka vytvářet z několika blízkých barevných bodů vjem jediného barevného bodu. Barva vnímaného bodu je pak do jisté míry průměrem skutečných bodů. Pro vytváření více barevných bodů z jednoho původního se používají ***metody náhodného rozptýlení, pravidelného (maticového) rozptýlení*** a také ***distribuce zaokrouhlovací chyby.***

2.1.2 Barevná paleta

Rozdíl mezi změnou barevného rozlišení u *šedotónového (grayscale)* obrazu[3] a barevného obrazu je v tom, že u barevného obrazu je třeba postupovat u každé jeho složky *(red, green, blue)* zvlášť. Druhým problémem pak u barevného obrazu je, jak nastavit vhodnou barevnou paletu. Lze použít už některou z přednastavených barevných palet, které přiřadí konkrétní rozsah barev ze zdrojového obrazu odpovídající barvě v cílovém obraze. Mnohem efektivnější ale je, když se zdrojový obraz nejdříve analyzuje a zjistí se, jaké odstíny barev zdrojový obraz obsahuje nejvíce.

Mezi **přednastavené palety** patří barevná ***paleta 3-3-2***, tato paleta kóduje výsledné barvy do 8 bitů, jelikož 8 není dělitelné třemi, tak je pro červenou a zelenou barvu použito po 3 bitech a pro modrou jen 2 bity. Lidské oko není příliš citlivé na modrou barvu, tudíž toto omezení tolik nevadí.

Druhou možností pak také je ***paleta přizpůsobená obrazu***. U této palety nedochází k tak citlivé ztrátě informace. Nejdříve je vždy třeba obraz analyzovat pomocí histogramu a takový histogram je pak vhodné reprezentovat v barevné krychli RGB. V této krychli je třeba nalézt oblasti s nejhustším obsazením barev a takovým oblastem pak přiřadit barvu z barevné palety.

2.2 Geometrické transformace diskrétního obrazu

Dvourozměrné geometrické transformace se dělí na lineární a nelineární transformace. Mezi lineární transformace patří posunutí, otočení, změnu měřítka, nebo zkosení a mezi nelineární pak patří warping a morfing.

Geometrické transformace pracují buď s bodem, nebo předpokládají spojitou reprezentaci objektu. Diskrétní obraz je jiné povahy, a to s sebou nese určité problémy. Pokud se bude transformovat obraz A na obraz B, tak obecná geometrická transformace přiřazuje pixelu, který má diskrétní souřadnice *[i,j]* nějaké neceločíselné místo *[x,y]*. V novém obraze pak mohou vznikat „díry“, nebo několik pixelů se může mapovat na jedno místo.

---

[3] Šedá je také barva ☺

2.2.1 Lineární geometrické transformace obrazu [13]

- ***Posunutí obrazu*** neboli translace je definováno maticí transformace:

$$T(X_t, Y_t) = \begin{bmatrix} 1 & 0 & X_t \\ 0 & 1 & Y_t \\ 0 & 0 & 1 \end{bmatrix}$$

- ***Změna měřítka*** ovlivňuje současně polohu i velikost transformovaného objektu ve směru souřadnicových os. Transformační matice je:

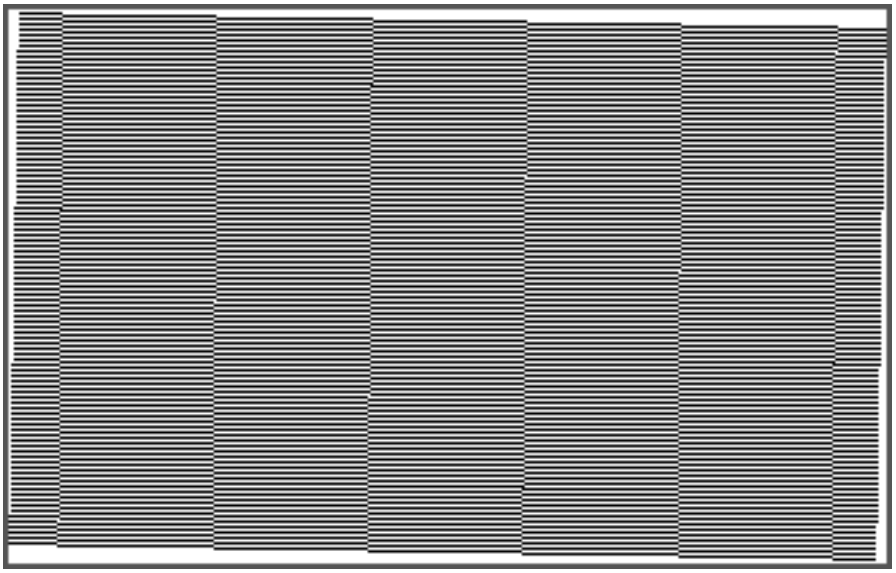
$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- ***Otočení obrazu*** o libovolný úhel představuje transformaci:

$$x' = x \cdot \cos\alpha - y \cdot \sin\alpha,$$
$$y' = x \cdot \sin\alpha + y \cdot \cos\alpha,$$

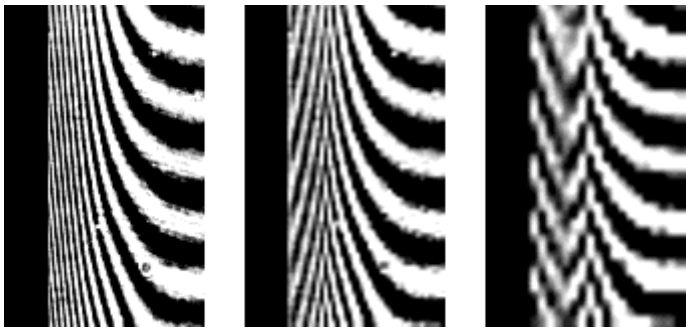
kde *x, y* jsou souřadnice původního bodu, x', y' souřadnice otočeného bodu a α je úhel otočení.

Otočení na nespojitých digitálních obrazech může způsobovat jev zvaný *alias*, který vzniká nepřesným přepočtem souřadnic bodů při aplikaci funkce otočení a díky rastru obrazových bodů, viz obrázek č.6. Ve výsledném obraze tak mohou vznikat efekty, které se v původním obraze nevyskytovaly. Po inverzní transformaci navíc nemusí vzniknout původní obraz.



Obr. 6: Otočení obrazu s horizontální osnovou proužků o malý úhel

- **Změna měřítka** obrazu znamená přepočtení intenzit bodů v obraze v poměru daném výsledným rozměrem obrazu *n* a původním rozměrem obrazu *m*. Intenzita výsledného *i-tého* bodu *b[i]* není závislá pouze na intenzitě *a[i]* jednoho původního *i-tého* bodu obrazu, ale je složena z intenzit více blízkých bodů. Přepočet intenzit je třeba provést dvakrát, a to v horizontálním a vertikálním směru.



Obr. 7: Moaré obrazce při

nedostatečném počtu obrazových bodů

Změna měřítka obrazu umožní získat např. obraz s větším počtem bodů, čímž se zvětší i počty bodů na zobrazení jednoho objektu v obraze. Podobně jako v případě otočení obrazu, může dojít i zde k jevu zvanému alias, který v tomto případě nedostatečným vzorkováním původního obrazu způsobí nežádoucí moaré obrazce v místech hustých proužků, viz obrázek č. 7.

2.2.2 Warping a morfing

Warping obrazu, česky *kroucení*, *zvlnění, pokřivení* či *deformace*, se provádí aplikací nějaké nelineární transformace na jediný obraz, který se mění jako v křivém zrcadle. Jako ***morfing*** se označuje obecný proces, kdy se jeden obraz postupným přechodem mění v jiný. Morfing se dělí na dva druhy. Jednodušší morfing *(stop-motion)* spočívá ve spojení obrazů, které obsahují nepohyblivé objekty [1]. Složitější variantou pak je *go-motion* morfing. To je takový morfing, kdy na sebe navazují obrazy, které jsou vlastně sousledností pohybujících se objektů. Pro správný morfing musí mít oba objekty stejný počet uzlů, hran a plošek.

Postup morfingu je následující: ve zdrojovém obraze se zvolí několik uzlových bodů, s tím, že více bodů znamená větší přesnost, ale i větší výpočetní náročnost. V cílovém obraze se musí nastavit polohy odpovídajících uzlových bodů podle zdrojové strany. Následně se zvolí počet kroků pro transformaci (*morfing*). Nutnou podmínkou je, aby se trasy přeměny jednotlivých bodů mezi sebou nekřížily.

Toto je *bodový morfing*, dalšími možnostmi jsou *úsečkový* nebo *síťový warping*. Princip warpingu je podobný, ale s tím rozdílem, že zdrojový a cílový obraz je stejný.

2.3 Algoritmy pro lokální úpravy v obraze

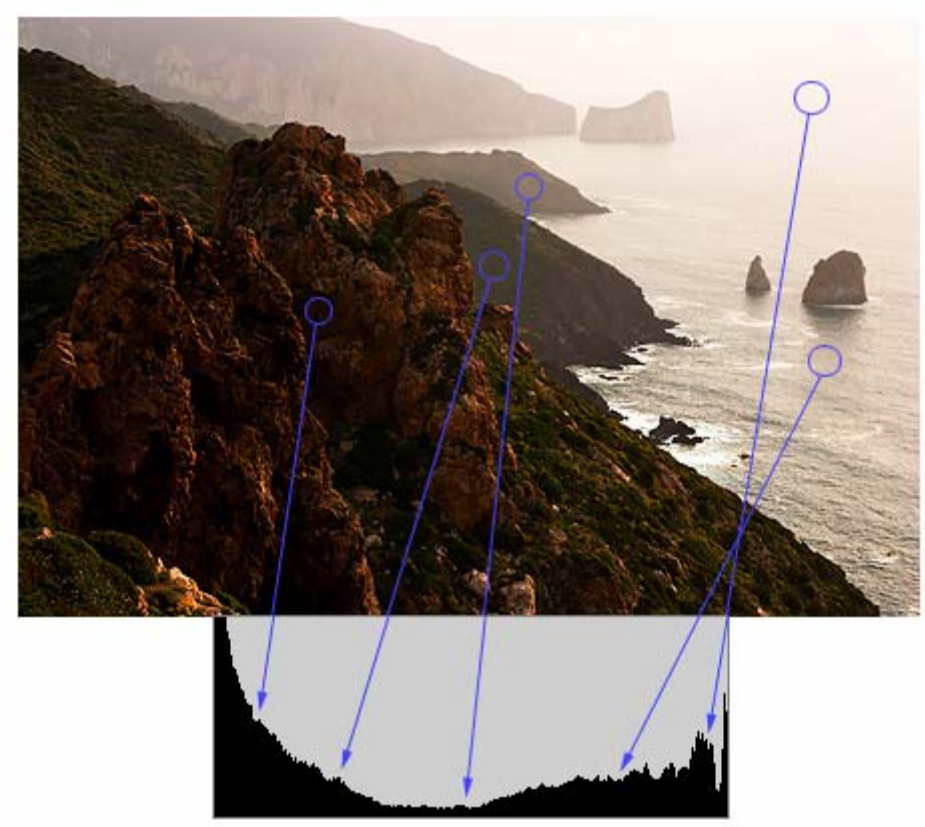
Při úpravách kvality obrazů jsou velmi důležité algoritmy pro lokální úpravy obrazů. Tyto algoritmy mohou odstraňovat drobné defekty v obrazech, které vznikají např. nečistotami či odlesky v sestavě při záznamu obrazů, vadami optických prvků, nedokonalostí zobrazovaných objektů, malým počtem obrazových bodů apod. Odstraňování některých drobných defektů v obraze lze sice provádět i automaticky (např. odstraňování zrnitosti), ale větší a ojedinělé defekty se odstraňují nejlépe interaktivně **retušováním**.

Pro retušování defektů lze využít metodu absolutní změny intenzity, relativní změny intenzity, retušování metodou razítkování a retušování metodou rozmazávání. Provádí-li se retušování pomocí myši, vztahuje se úprava obrazu pouze na jisté okolí bodu, na který myš ukazuje. Toto okolí bodu může mít různou velikost i tvar, přičemž nejčastěji se jedná o symetrická okolí bodu kruhová, eliptická či čtvercová [13]. Úprava intenzity

obrazových bodů v daném okolí se děje obvykle s jistou váhou danou maskou s Gaussovským rozložením váhových koeficientů.

3 ALGORITMY PRO ÚPRAVU OBRAZU VYUŽÍVAJÍCÍ HISTOGRAM

Je důležitým klíčem k charakterizaci obrazu. Kvantifikuje množství a frekvenci barev obsažených v obraze. Je neocenitelným pomocníkem při úpravách kvality obrazu. Pomocí informací z histogramu je možné u obrazu snížit, či zvýšit jas a kontrast. Dále umožňuje nastavit vhodný práh, provést gama korekci, vytvořit negativ obrazu, či změnit počet barev. V neposlední řadě ekvalizace histogramu dokáže během chvilky zásadně vylepšit obraz. V kapitole 7 jsou popsány vlastní algoritmy, které jsou použity v programu Imagine.



Obr. 8: Obraz a vazby na histogram[4]

[4] obrázek převzat z http://www.cambridgeincolour.com/tutorials/histograms1.htm

Z grafické podoby histogramu lze usuzovat nejen na stav digitálního obrazu, ale také na možnosti úpravy obrazu. Histogram je grafické znázornění vektoru s počtem složek rovným počtu možných úrovní intenzit obrazových bodů. Tento vektor je označen písmenem *H* a hodnota každé složky vektoru odpovídá četnosti bodů příslušné intenzity v obraze. Klasifikace obrazu pomocí histogramu rozlišuje čtyři základní druhy obrazů. *Jasný obraz* - ***high-key*** má převážnou většinu barev přítomnou ve světlech. Jeho opakem je tmavý obraz označovaný jako ***low-key*** [1].

Obr. 9: Histogramy, vlevo tmavý obraz - low-key,

vpravo pak jasný obraz - high-key

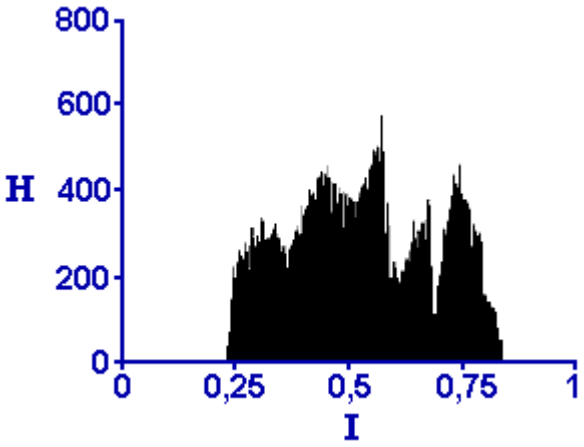
Středotónový obraz – ***mid-key***, má většinu barev zastoupenou kolem střední hodnoty. Poslední typem obrazu je obraz s vysokým kontrastem. Velikost kontrastu je dána mírou rozdílu mezi středními hodnotami na jedné straně a světly a stíny na straně druhé. Z tohoto důvodu se také středotónový obraz označuje jako obraz s nízkým kontrastem. Histogram, ve kterém se nacházejí dva ostré vrcholy se nazývá *bimodální*.

Obr. 10: Vlevo je histogram neostrého obrazu - mid-key, vpravo pak histogram obrazu s vysokým kontrastem.

3.1 Ekvalizace histogramu

Ekvalizace histogramu je algoritmus, který změní rozložení intenzit v obraze tak, aby se v něm vyskytovaly pokud možno intenzity v širokém rozmezí, a to přibližně se stejnou četností. U obrazů s konečným počtem obrazových bodů se lze tomuto cíli jen

přiblížit, viz obrázek č.12.[5] Ekvalizace umožňuje v obraze s celkově vysokým kontrastem zvýraznit špatně rozpoznatelné detaily s nízkým kontrastem. Typický tvar histogramu digitálního obrazu, který nevyužívá všechny dostupné úrovně intenzit, je uveden na obrázku č.11. Z histogramu je zřejmé, že v obraze schází zejména hodnoty intenzit na okrajích intervalu <0,1>, což obvykle snižuje kvalitu obrazu.



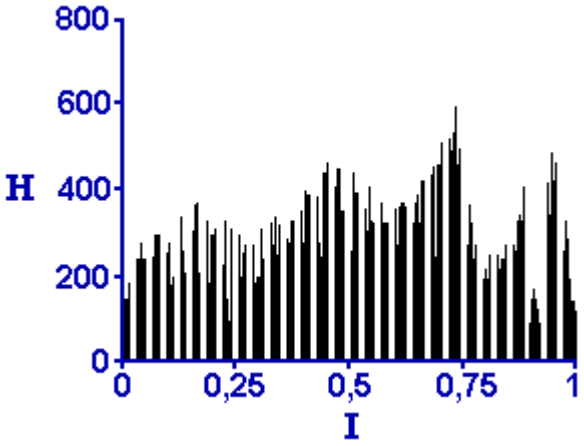
Obr. 11: Histogram obrazu, který nevyužívá všechny úrovně intenzit

Výsledná intenzita obrazového bodu po ekvalizaci I' se vypočítá z původní intenzity obrazového bodu I, a to pomocí vztahu [13]:

$$I' = \frac{I}{X \cdot Y} \cdot \sum_{i=I_0}^{i=I} H(i)$$

kde $H(i)$ je *i-tá* složka vektoru histogramu (počet bodů v obraze se stejnou intenzitou), X, Y jsou rozměry obrazu a I_0 je nejnižší intenzita původního obrazu.

[5] Obraz po aplikaci ekvalizace histogramu v programu Imagine, je uveden v kapitole 7.3.5



Obr. 12: Histogram obrazu

po ekvalizaci

3.2 Kontrast a jas

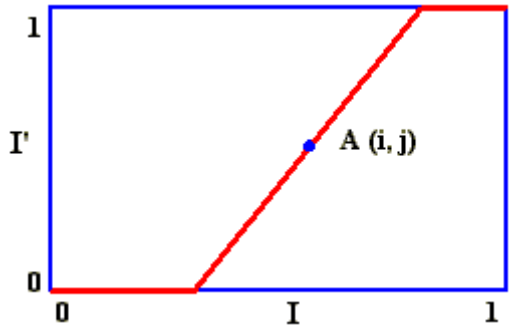
Nejčastěji používanou funkcí při úpravách kvality obrazů je ***kontrast a jas***. Následný text je zaměřen na šedotónové obrazy s intenzitou *I* od 0 do 1. U barevných obrazů, je třeba algoritmus aplikovat na všechny tři barevné složky (červenou, zelenou a modrou). Funkce změny jasu umožní upravovat příliš tmavé, či příliš světlé obrazy, na obrazy s jasovou úrovní vhodnou pro další vyhodnocování. Změna kontrastu umožní zvětšit kontrast objektů v obraze. Intenzitu *I'* každého upravovaného bodu lze získat ze vzorce:

$$I' = j + k \cdot (I - i)$$

kde *I* je intenzita původního obrazového bodu, *j* je jasová složka (implicitně *j = 0,5*), *k* je složka kontrastu (implicitně *k = 1*) a hodnota *i* bývá obvykle konstantní a má hodnotu

i = 0,5. Po aplikaci tohoto vztahu se hodnota intenzity I' upraví tak, že všechny hodnoty větší než 1 se zarovnají na 1 a všechny hodnoty menší než 0 se zarovnají na 0, viz také obr. č.13. [6]

[6] Obrazy po změně kontrastu a jasu v programu Imagine jsou uvedeny v kapitole 7.3.3



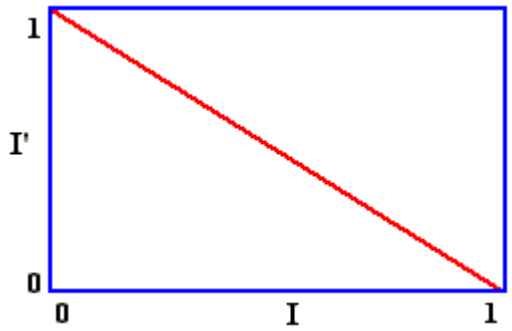
Obr. 13: Grafické znázornění

funkce kontrast a jas

3.3 Negativ

Funkce ***negativ*** se používá po skenování negativních předloh, před skládáním obrazů nebo také v souvislosti s funkcemi typu *eroze*, *dilatace* apod. Výsledná intenzita bodu negativního obrazu I' se získá z původní intenzity I dle obr. č. 14 nebo také dle vztahu:

$$I' = 1 - I$$



Obr. 14: Grafické znázornění

funkce negativ

3.4 Prahování

Funkce prahování usnadní v mnoha případech proces identifikace objektů v obraze. Pokud jsou intenzity I původního obrazu menší než zvolená prahová hodnota P, přiřadí se jim intenzita 0, v opačném případě intenzita 1.

Tato funkce může pracovat i s více prahy, přičemž u každého prahu dojde ke změně výsledné intenzity z hodnoty 0 na 1 či opačně. Prahování s více prahy, které rozdělují oblast intenzit *I* rovnoměrně je označováno jako *interferogram*.

3.5 Gama korekce

Dnes patrně dosluhující vakuové obrazovky, označované jako CRT, mají nelineární jasovou odezvu intenzity fosforů na stínítku v závislosti na vstupním napětí katody. Křivka, která tuto nelinearitu charakterizuje, zhruba odpovídá mocnině 2.5. jinými slovy řečeno, pokud je potřeba zobrazit intenzitu i, ve skutečnosti se zobrazí $i^{2.5}$. Intenzita napětí je v rozsahu od nuly do jedné a tak tato nelinearita obraz ztmavuje. Podstatnější vlastností je, že odezva není lineární. Samozřejmým požadavkem je, aby intenzita 0.5 byla dvojnásobkem intenzity 0.25 a tak je nutné se s touto vlastností nějak vypořádat.

Operace, která tuto nelinearitu odstraňuje, se jmenuje gama korekce a vyhledávací tabulka této funkce se vypočítá podle vztahu:

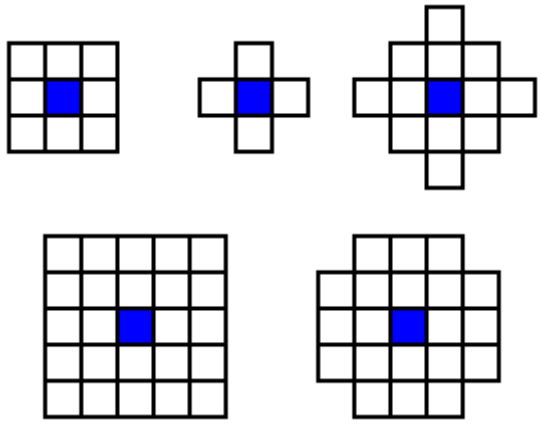
$$i' = i^{\frac{1}{\gamma}}$$

Hodnota γ je závislá na typu obrazovky. Většina monitorů, a stejně tak i televizních obrazovek, má tuto konstantu předem nastavenou. Kvalitnější monitory se za pomocí speciálního čidla mohou kalibrovat automaticky, protože hodnota zkreslení závisí kromě jiného i na teplotě. Hodnota bývá $\gamma = 2.5 \pm 0.3$ [1]

Gama korekce je nezbytná hlavně kvůli tomu, aby obraz při změně zobrazovacího zařízení vypadal stejně.

4 FILTRACE OBRAZŮ

Při úpravách kvality obrazů jsou důležité různé typy filtrace. Tyto filtrace mohou odstranit či zvýraznit zrnitost záznamu nebo zvýraznit různé jiné objekty v obraze. Jedná se vlastně o úpravu intenzity obrazového bodu s vazbou na jeho okolí, přičemž okolí bodu může mít různý tvar, viz příklady uvedené na obrázku č.15. Jsou zde uvedena symetrická okolí bodu, kdy upravovaný bod (vybarvený) je uprostřed svého okolí. Kromě symetrických okolí existují i nesymetrická okolí bodu a adaptivní okolí bodu.



Obr. 15: Příklady různých typů

symetrických okolí bodu

Filtry lze aplikovat buď na celý obraz, nebo jen na dočasně vymezenou oblast obrazu. Hrany objektů lze zachovat vhodnou definicí oblastí pro filtraci.

Filtry rozdělujeme na ***lineární*** a ***nelineární***. Lineární filtry lze dále dělit na filtry typu ***dolní propust a horní propust*** [13]. V kapitole 7 jsou popsány vlastní algoritmy, které jsou použity v programu Imagine.

4.1 Lineární filtry

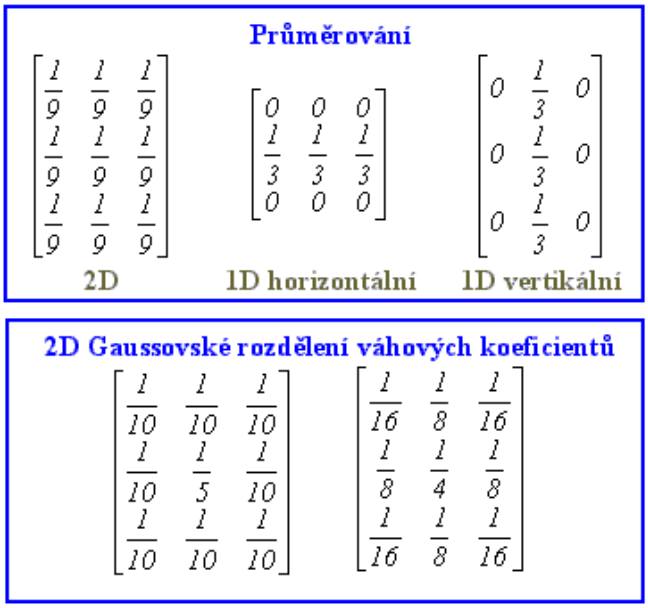
U lineárních filtrů je intenzita upravovaného bodu rovna součtu součinů intenzit bodu v okolí a příslušných koeficientů z matice váhových koeficientů.

Tyto filtry pracují tak, že při procházení celého obrazu nahrazují hodnoty zpracovávaného pixelu lineární kombinací pixelů ležících v jeho okolí. děje se tak pomocí tzv. ***konvoluční matice*** $C = (c_{i,j})$, což je čtvercová matice řádu 2n+1. Tyto filtry se nazývají lineární, protože novou hodnotu zpracovávaného pixelu dostaneme tzv. lineární kombinací hodnot

pixelů v jeho okolí. K tomu aby lineární filtr fungoval odpovídajícím způsobem, je třeba vhodně nastavit konvoluční matici. Vhodnou volbou lze získat filtry nejrůznějších vlastností. Základní dvě skupiny filtrů jsou filtry dolní a horní propust.[7]

4.1.1 Dolní propust

Lineární filtry typu ***dolní propust*** slouží především k odstranění vysokých prostorových frekvencí intenzit v obraze, čímž dojde k potlačení nežádoucího šumu, ale také k potlačení detailů v obraze. Tyto filtry se používají především k odstranění zrnitosti v obrazech. Příklady matic váhových koeficientů (s okolím 3 x 3 bodů) pro typické filtry typu dolní propust jsou uvedeny na obrázku č.16. Patří mezi ně matice váhových koeficientů pro tzv. průměrování ve dvou směrech, ***průměrování ve směru horizontálním*** a ***průměrování ve směru vertikálním***. Dále jsou zde uvedeny matice váhových koeficientů s Gaussovským rozložením hodnot. Pro filtry typu dolní propust je charakteristické, že součet váhových koeficientů v matici je roven 1. Pokud by součet byl vyšší, tak by filtrace obraz zesvětlovala a naopak, jádro se součtem nižším jak 1, by obraz při filtraci ztmavovalo.



Obr. 16: Příklady matic váhových koeficientů

[7] Obrázek po aplikaci lineárního filtru v programu Imagine je uveden v kapitole 7.3.1

pro filtry typu dolní propust

4.1.2 Horní propust

Lineární filtry typu ***horní propust*** umožní zvýraznit detaily v obraze, ale zároveň také dojde obvykle ke zvýraznění šumu. Pro filtry typu horní propust je charakteristické, že součet váhových koeficientů v matici je roven 0, viz např. matice váhových koeficientů uvedená na obrázku č.17.

Váhové koeficienty 2D filtru typu horní propust

$$\begin{bmatrix} -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \\ -\frac{1}{9} & \frac{8}{9} & -\frac{1}{9} \\ -\frac{1}{9} & -\frac{1}{9} & -\frac{1}{9} \end{bmatrix}$$

Obr. 17: Příklad matice váhových koeficientů

pro filtr typu horní propust

4.2 Nelineární filtry

Nelineární filtry nepočítají intenzitu upravovaného bodu, ale vybírají z okolí vhodnou hodnotu, kterou pak dosazují do upravovaného bodu. Oproti lineárním filtrům mají tu výhodu, že nepřidávají do obrazu žádnou novou hodnotu intenzity. [8]

4.2.1 Filtr minimum

Filtr ***minimum***, označovaný také jako ***eroze***, vybírá z blízkého okolí bod s minimální hodnotou intenzity a tu dosadí do upravovaného bodu[5]. Umožňuje erozi tmavých proužků na světlém pozadí či dilataci světlých proužků na tmavém pozadí. Může sloužit k potlačení šumu ve světlé části obrazu, nebo k zeslabení čar ve schématech.

[8] Obrázek po aplikaci nelineárního filtru v programu Imagine je uveden v kapitole 7.3.2

4.2.2 Filtr medián

Filtr ***medián*** vybírá z blízkého okolí bod se střední hodnotou intenzity, kterou pak dosadí do upravovaného bodu. Tento filtr je velmi účinný pro potlačení šumu a lze jej s výhodou používat např. pro vyhlazování zrnitosti. Jeho nevýhodou je, že ohlazuje hrany objektů, čímž mění jejich tvary.

4.2.3 Filtr maximum

Filtr ***maximum***, označovaný také jako ***dilatace***, vybírá z blízkého okolí bod s maximální hodnotou intenzity, kterou pak dosadí do upravovaného bodu. Umožňuje dilataci tmavých proužků na světlém pozadí nebo erozi světlých proužků na tmavém pozadí. Může sloužit k potlačení šumu ve tmavé části obrazu, nebo také k zesílení čar ve schématech.

5 CHARAKTERISTIKA PROGRAMOVACÍHO JAZYKA JAVA

Java je vyspělý programovací jazyk, obsahující všechny vlastnosti, které jsou vyžadovány v moderním programování, od modularity programu, řídících konstrukcí, přes silnou typovou kontrolu, multithreading, ošetření výjimek, správu paměti, i silnou podporu pro databáze, XML a síťové operace. K jejím výhodám kromě multiplatformity, patří i robustnost, škálovatelnost a vysoká bezpečnost, která jí profiluje i pro používání na kritické aplikace na mainframeových počítačích [11].

5.1 Porovnání s ostatními jazyky

5.1.1 Vlastnosti Javy

- **jednoduchý –** jeho syntaxe je zjednodušenou (a drobně upravenou) verzí syntaxe jazyka C a C++. Odpadla většina konstrukcí, které způsobovaly programátorům problémy a na druhou stranu přibyla řada užitečných rozšíření.
- **objektově orientovaný –** s výjimkou osmi primitivních datových typů jsou všechny ostatní datové typy objektové.
- **distribuovaný –** je navržen pro podporu aplikací v síti (podporuje různé úrovně síťového spojení, práce se vzdálenými soubory, umožňuje vytvářet distribuované klientské aplikace a servery).
- **interpretovaný –** místo skutečného strojového kódu se vytváří pouze tzv. mezikód – ***bytecode***. Tento formát je nezávislý na architektuře počítače nebo zařízení. Program pak může pracovat na libovolném počítači nebo zařízení, který má k dispozici ***interpret Javy*** - virtuální stroj Javy.
- v posledních verzích Javy není bajtový kód jenom interpretován, ale častěji prováděné části (např. cykly nebo často volané metody) jsou před prvním svým provedením dynamicky zkompilovány do strojového kódu daného počítače (tzv. just in time compilation, JIT). Tato vlastnost zásadním způsobem zrychlila provádění programů v Javě.
- **robustní –** je určen pro psaní vysoce spolehlivého softwaru – z tohoto důvodu neumožňuje některé programátorské konstrukce, které bývají častou

příčinou chyb (např. správa paměti, příkaz goto, používání ukazatelů). Používá tzv. silnou typovou kontrolu – veškeré používané proměnné musí mít definovaný svůj datový typ.

- **bezpečný** – má vlastnosti, které chrání počítač v síťovém prostředí, na kterém je program zpracováván, před nebezpečnými operacemi nebo napadením vlastního operačního systému nepřátelským kódem.
- **nezávislý na architektuře** – vytvořená aplikace běží na libovolném operačním systému nebo libovolné architektuře. Ke spuštění programu je potřeba pouze to, aby byl na dané platformě instalován správný virtuální stroj. Podle konkrétní platformy se může přizpůsobit vzhled a chování aplikace.
- **přenositelný** – vedle zmíněné nezávislosti na architektuře je jazyk nezávislý i co se týká vlastností základních datových typů (je například explicitně určena vlastnost a velikost každého z primitivních datových typů).
- **víceúlohový** – podporuje zpracování vícevláknových aplikací (multithreading)
- **dynamický** – Java byla navržena pro nasazení ve vyvíjejícím se prostředí. Knihovna může být dynamicky za chodu rozšiřována o nové třídy a funkce, a to jak z externích zdrojů, tak vlastním programem.
- Nižší rychlost, způsobená zpracováním v runtime prostředí, může být urychlena s pomocí specializovaných překladačů na cílovém prostředí (Java just-in-time, JIT)[9]. Tyto překladače vytvářejí kód pro konkrétní operační systém, tímto se dosahuje zvýšení výkonu aplikace a zamezí se případnému zpětnému inženýrství programu.

[9] Například překladač JET 4.1 Professional, který zároveň umožňuje i ochranu kódu. K dispozici na http://www.xlsoft.com/en/products/jet/index.html

5.1.2 Porovnání výkonu s programy v C#

Jazyk C# se v mnohém podobá jazyku Java. Přesto je jeho orientace trochu odlišná. Zatímco Java se před několika lety prezentovala jako zajímavá alternativa k vytváření běžných aplikací, a přestože se v mnoha oblastech velmi osvědčila, nikdy se nestala nástrojem většiny programátorů. Jazyk C# naproti tomu míří do mainstreamu. Lze s ním vytvářet jak „obyčejné“ aplikace, tak webové služby nebo aktivní webové stránky.

Nejde o žádné teoretické matematické výpočty, ale o příklady ze skutečných aplikací. [12]

Název testu	C# (doba trvání u Javy = 100%)
Počítání řádků, slov a znaků	62,5%
Zpracování výjimek	176,82%
Hash funkce	215,80%
Násobení matic	181,50%
Volání metod	50,80%
Erathostenovo síto	177,70%

100% - základ – doba trvání v Javě.

50% - tento výsledek by znamenal, že C# je dvakrát rychlejší než Java.

200% - by znamenalo, že Java je dvakrát rychlejší než C#.

Byly vybrány funkce, které se poměrně často používají. Důležitý je výsledek u násobení matic, který naznačuje rychlost zpracování obrazových informací.

5.2 Programátorská podpora Javy

Java hraje určitou roli konkurenta a soupeře společnosti Microsoft a také programovacím jazykům C++, či C#. Zatímco ale C++ je určen pro klasické programy a C# pro webové aplikace, tak Java je s úspěchem používána při vývoji klasických programů, programu určených pro web, i programů pro mobilní telefony.

Pro kompilaci zdrojových kódů je potřeba vývojářský balík, nazvaný Java Development Kit (JDK). Kromě "originálního" JDK od Sun Microsystems existují ještě další verze od jiných producentů

Dokumentace k nástrojům JDK a ke knihovnám Javy není součástí instalačních balíků JDK, instaluje se zvlášť. Balík je poměrně objemný (pro JDK 1.5.0 má 45 MB, po rozbalení na disku zabere několikanásobek, většinou přes 250 MB).

Jak provozní, tak vývojové prostředí jsou k dispozici pro celou řadu platforem. Potěšující je, že Javu 5.0 je možné používat i na 64-bitových systémech (v 64-bitovém režimu) a využít tak jejich plnou sílu.

5.2.1 Kategorie Javy

Java ME - Java Platform, Micro Edition (dříve Java 2 Micro Edition nebo J2ME) je velmi malá platforma Java. Takto omezená sada virtuálního stroje a API umožňuje vytvářet a spouštět programy určené pro zařízení s malým výkonem, jako je mobilní telefon, PDA, apod.

Java SE - Java Platform, Standard Edition (dříve Java 2 Standard Edition nebo J2SE). Tato platforma je nejpoužívanější a slouží k vývoji běžných programů, nejčastěji pro PC.

Java EE - Java Platform, Enterprise Edition je standardem pro vývoj přenosných, robustních, škálovatelných a bezpečných serverových Java aplikací. Na základě podpory Java SE, může Java EE nabízet webové služby, modelové komponenty, správu a komunikaci API prostředí [6].

5.2.2 Zdroje informací o Javě

- Jako nejvhodnější způsob, jak se naučit Javu, je si koupit učebnice o Javě. Mezi ty nejlepší patří knihy Pavla Herouta [2], [3] a [4], který pracuje jako odborný asistent na katedře informatiky a výpočetní techniky Západočeské univerzity v Plzni.
- Stránky studentského informačního serveru západočeské univerzity, které obsahují základní informace o Javě - http://dione.zcu.cz/java
- Český server, který se věnuje výhradně programovacímu jazyku Java: http://java.cz/

- V angličtině pak také na: http://www.sun.com
- Ale absolutně nejlepším způsobem jak během programování vyhledávat informace je si stáhnout dokumentaci z http://www.allimant.org/javadoc

  Jedná se o nápovědu, referenční příručku, dokumentaci a nekonečně obsáhlý seznam stránek a publikací o Javě. Tato nápověda obsahuje odkazy na informační zdroje přímo z firmy Sun, jedná se tedy o velmi kvalitní a určitě kompletní zdroj informací o Javě. Pro programátora, který umí anglicky se zcela jistě jedná o neocenitelný zdroj informací.

Java se stává v poslední době velmi populárním programovacím jazykem a tudíž existuje nepřeberné množství dalších serverů, které se zabývají programováním v tomto jazyce.

5.3 Vývojová prostředí pro Javu

V současné době existuje mnoho aplikací určených k vývoji programů v jazyce Java. Podstatná většina z nich je podporována open-source komunitami a jsou tudíž většinou zdarma. Programy lze psát v libovolném editoru[10] a kompilovat z příkazové řádky. Všeobecně se však používají integrovaná vývojová prostředí [14]

5.3.1 NetBeans[11]

Open-source projekt původně vyvíjený českými autory, později prodaný firmě ***Sun Microsystems***, která z něj učinila základ pro svá vývojová prostředí. Pro většinu použití však "základní" NetBeans bohatě stačí. Sám program je napsán v Javě, uživatelské rozhraní používá knihovnu Swing ze standardního balíku Javy. Prostředí využívá jako aplikační základ NetBeans Framework.

- Umožňuje snadnou editaci, kompilaci a spouštění programů
- Má komfortní funkce pro debugging.
- Má grafického návrháře UI.

[10] Například i v poznámkovém bloku.

[11] V současné verzi NetBeans 5.0 byl vytvořen i program Imagine.

- Obsahuje řadu různých pomocných nástrojů (internacionalizace, práce s dokumentací, automatické aktualizace apod.), další lze přidat jako pluginy.
- Nevýhoda spočívá v tom, že spotřebuje hodně paměti a mnoho času procesoru a je také poměrně nestabilní[12].

5.3.2 Eclipse

Opět open-source projekt, ovšem podporovaný firmou IBM, která na něm staví své *WebSphere* Studio. Eclipse je v podstatě velmi obecný „*interface*“, do kterého se funkcionalita přidává pomocí pluginů - existuje jich velké množství a další stále vznikají.

- Běží opět v Javě, ale jako GUI místo Swingu používá SWT (částečně nativní implementace).
- Vývojové prostředí jako takové tvoří základní sada pluginů, další lze přidávat.
- V základní sadě poněkud méně funkcí než NetBeans.
- Chybí grafický návrh GUI (lze přidat pluginem, ale nekomerční plug-in je velice špatný)
- Nevýhoda je opět ve velké paměťové náročnosti, ale je o něco rychlejší než NetBeans, mnohem stabilnější a není třeba jej instalovat.

5.3.3 BlueJ

Společný projekt několika univerzit určený jako prostředí pro výuku Javy. Je to sice „*closed-source“* program, ale je zdarma.

- Je navržen k výuce a je tedy velmi vhodný pro začátečníky.
- Obsahuje grafický, abstraktní, čistě objektový návrh dat.
- Běží v Javě bez závislosti na platformě.
- Rozšiřitelný pomocí pluginů.

---

[12] Bohužel, vlastní zkušenost autora této práce ☹

5.3.4 JBuilder

Vývojový balík od firmy Borland, dostupný v několika variantách, jedna z nich je bezplatná. Běží opět v Javě

- Množina funkcí velice podobná jako u NetBeans, spíše ještě širší

Všechna tato prostředí fungují v operačních systémech Windows i Linux, proto je možné zvolit kterékoliv z nich. Doporučit některé z nich je těžké, záleží na osobním vkusu. Nejlepší cestou asi bude si každé z prostředí vyzkoušet a pak se rozhodnout. Jako nejlepší se může zdát Eclipse, ten má ale handicap díky špatnému grafickému manažeru GUI , takže je vhodný pro programy, které buď nevyužívají GUI, nebo jen velmi málo. Eclipse také přijde vhod těm, kteří mají tu trpělivost si potřebné pluginy nainstalovat. Výhoda eclipse je i ta, že jej není třeba instalovat. Na programy s náročnějším GUI se nejvíce hodí NetBeans. Nutným minimem k programování Javy je pak i 1GHz procesor a alespoň 256 MB RAM.

5.4 Tvorba grafického uživatelského rozhraní pomocí Swing

Cílem původního návrhu knihovny grafického uživatelského rozhraní (GUI), používaného v jazyce Java verze 1.0 bylo umožnit programátorovi sestavovat takové GUI, které bude vypadat slušně na všech platformách. Ale knihovna - *abstract windows toolkit* - *AWT*, je schopna fungovat na všech platformách, ale její vzhled je pouze průměrný a navíc je velmi restriktivní. Lze používat pouze 4 druhy písma a není přístup k propracovanějším prvkům GUI existujícím v hostitelském operačním systému[8].

Situace se zlepšila s příchodem událostního modelu Javy AWT 1.1. Ten je mnohem přehlednější, objektově orientovaný a navíc využívá JavaBeans – model komponentového programování, navržený pro snadnou tvorbu vizuálního programovacího prostředí. Java 2 pak kompletně nahrazuje původní model Java 1.0 AWT, novou knihovnou *Java Foundation Classes* - *JFC*, jejíž část GUI je nazývána "***Swing***". Je to knihovna snadno použitelných a srozumitelných objektů *JavaBeans*, které lze programovat pomocí uživatelsky přívětivého grafického rozhraní.

Mezi komponenty knihovny Swing patří tlačítka, skupiny tlačítek, ikony, plovoucí komentáře, textová pole, okraje, posuvná podokna, minieditor, zaškrtávací políčka,

přepínače, pole se seznamem, seznamy, karty, okna zpráv, nabídky, kreslení, dialogová okna, dialogová okna pro manipulaci se soubory, stromy, tabulky a schránka.

5.5 Aplety

Aplety jsou jednoduché programy, spuštěné uvnitř prohlížeče www. Vzhledem k tomu, že musí být bezpečné, jsou jejich možnosti poněkud omezené. Aplety jsou však i přesto velmi výkonným nástrojem [3].

Hlavním omezením je, že aplet nemůže mít přístup k obsahu pevného disku lokálního počítače. Toto lze změnit, pokud se v nastavení operačního systému umožní přístup k disku důvěryhodným apletům.

Na druhou stranu mezi jednoznačné výhody patří, že aplet není třeba instalovat a zároveň uživatel nemusí mít strach, že by apletem infiltroval vir do systému [8].

Díky těmto vlastnostem se aplety používají hlavně pro intranetové aplikace typu klient/server

5.6 Knihovna java.awt.Image [15]

Poskytuje třídy pro tvorbu a úpravy obrazů. Obrazy jsou vytvářeny různými objekty, které také ovlivňují typ ***ImageProducer***, vlastnosti obrazových filtrů a ***ImageConsumer***. Tato knihovna umožňuje renderovat obrazy zatímco jsou nahrávány, nebo vytvářeny. Dále pak lze kdykoliv smazat paměť a obnovit ji.

Součástí Javy je už od verze JDK 1.0

Knihovna *java.awt.image* obsahuje ve verzi Javy JDK 5.0 celkem 8 rozhraní a 42 tříd pro práci s grafikou.

Mezi nejdůležitější třídy a rozhraní patří:

5.6.1 Třída BufferedImage

Nejvhodnější způsob pro práci s grafikou v Javě. Její rodičovská třída *Image* neumožňovala mnoho funkcí, které se pak pracně museli programovat. A hlavně při vykreslování dat z třídy *Image* se projevovalo značné zpoždění.

Tato třída popisuje abstraktní třídu *Image* s přístupným zásobníkem obrazových dat, který dokáže velice rychle zpřístupnit potřebná data. Základním parametrem této třídy je typ barevného prostoru, ve kterém je uložen obraz. Jsou podporovány binární, indexové, grayscale, RGB, ARGB, nebo i uživatelské barevné modely. Důležité metody této třídy mají například funkce pro zjištění rozměrů obrazu, nebo změny hodnot v obraze, zjištění barevného modelu, který obraz používá. Obsahuje funkci *getRGB()*, která vrací vektor s hodnotami ve typu *int*. Tím, že je vektor typu *int* je ale omezena velikost dat, které lze touto funkcí získat. V praxi je pak možné pracovat s obrázky s maximálním rozlišením okolo 1400 na 1100 pixelů.

Možným řešením by tak bylo číst data jinou metodou, která by ale byla mnohem komplikovanější. Dalším řešením by také mohlo být to, že by se načetla jenom část obrazu (tak velká, aby se ještě vešla do vektoru typu *int*) a ostatní části obrazu by se načetly do dalších vektorů. Poté by se veškeré vektory spojily do jednoho, který by byl typu *long*.[13]

5.6.2 Třída ColorModel

Tato abstraktní třída vyjadřuje metody pro reprezentaci pixelových hodnot obrazu a komponent, kterými jsou například *Red, Green, Blue* a *alfa* hodnoty průhlednosti.

5.6.3 Třída ConvolveOp

ConvolveOp slouží ke konvoluci obrazu, který je uložen v *BufferedImage*. Používá k tomu definované jádro – ***kernel***.

5.6.4 Třída Kernel

Definuje matici, která při filtraci určuje výslednou hodnotu pixelu. Hodnota je závislá na konkrétním pixelu a jeho okolí.

---

[13] Typ *int* má velikost 32 bitů a long 64 bitů. [2]

5.6.5 Třída PixelGrabber

Implementuje rozhraní *ImageConsumer*, které může být připojeno k objektům obrazu, pro získání části pixelů v obraze.

5.6.6 Rozhraní ImageConsumer

Rozhraní pro objekty, které mají vazbu na data v obraze skrze rozhraní .

5.6.7 Rozhraní ImageProducer

Rozhraní pro objekty, které mohou produkovat obrazová data.

5.7 Třída Graphics2D [15]

Vznikla rozšířením třídy Graphics a poskytuje mnohem sofistikovanější metody v úpravách geometrie obrazu, v souřadnicovém systému, ve správě barev a v úpravách textu. Tato třída využívá uživatelského prostoru (*User space*), ten umožňuje nezávislost na cílovém zařízení. Základním úkolem této třídy je vykreslovat – „renderovat“ objekt v obraze. Tímto objektem mohou být tvary, text nebo obrázek.

Třída Graphics2D obsahuje mnoho metod pro vykreslování těchto objektů, mezi ty základní patří:

draw(Shape s) - záleží na specifikaci konkrétního tvaru (Shape). U tvaru se dá nastavit velikost, barevná výplň a okraje tvaru.

drawImage(BufferedImage img, ...) – vykreslí objekt *img*, v dalších parametrech lze specifikovat pozici pro vykreslení, typ filtrace, nebo afinní transformaci, která bude provedena na obraz před vykreslením.

drawString(String s, ...) - vykreslí řetězec znaků, specifikovat lze pozici, velikost a font písma.

fill(Shape s) – použije na tvar konkrétní výplň.

rotate(...) – rotace podle úhlu, nebo i spojená s posunem

scale(...) - změna měřítka. Poměr je udáván ve směru os x a y.

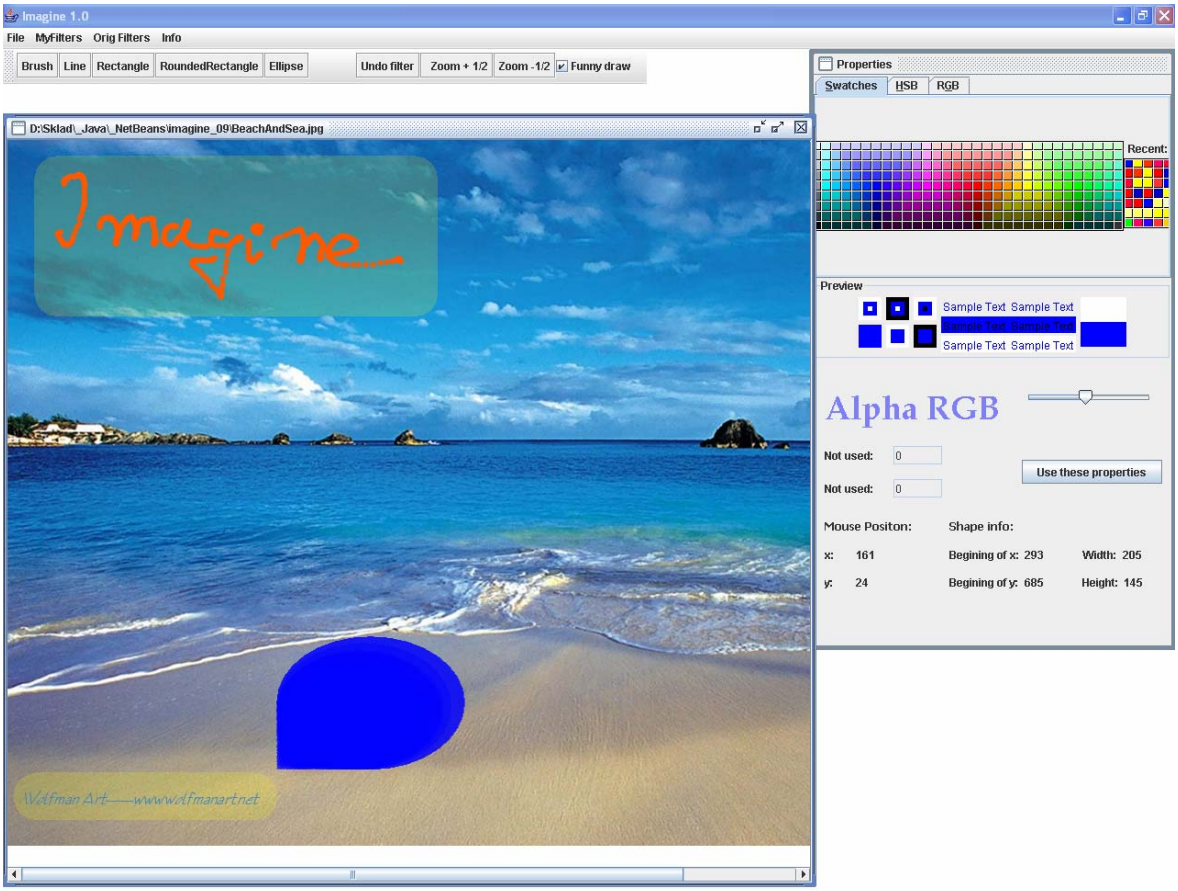
shear(...) - zkosení. Poměr je udáván ve směru os x a y.

translate(...) - posouvá objekt na novou pozici, zadanou pomocí souřadnic.

II. PRAKTICKÁ ČÁST

6 GRAFICKÝ EDITOR IMAGINE 1.0

Grafický editor Imagine 1.0 je program napsaný v programovacím jazyce Java. Umožňuje načíst, ukládat a také vytvářet bitmapové soubory s příponou BMP, JPEG a PNG. V programu lze kreslit štětcem, vytvářet základní geometrické tvary jako jsou čára, obdélník, zaoblený obdélník a elipsa. V programu je také možné si vybrat barvu pomocí palety barev, číselného zadání, nebo pomocí barevného pole, včetně jejich průhlednosti alfa.



Obr. 18: Program Imagine a ukázka jeho funkcí

6.1 Spuštění a ovládání programu

Hlavní funkcí programu jsou algoritmy pro úpravu obrazu, tak jak byly zmíněny v kapitolách 3 a 4. Algoritmy nevyužívají originálních „java“ algoritmů, i když by to práci programu výrazně zjednodušilo a zrychlilo. Smyslem této práce je totiž vytvořit vlastní algoritmy pro úpravy obrazů. Díky tomu je pak možné přesně měnit nastavení jednotlivých algoritmů.

6.1.1 Instalace a spuštění programu

Program není potřeba instalovat, je však potřené mít nainstalován J2RE. Pro OS Windows ve verzi 1.5.0 nebo vyšší. Nedoporučuje se také program spouštět na počítači s méně jak 64 MB operační paměti.

Program byl otestován na dvou počítačích.

První počítač: CPU AMD Sempron 2800+ (64bit), operační systém Fedora4 (kernel2.6.14) a J2RE ve verzi *jre-1_5_0_04-linux-amd64* (pro 64 bitové aplikace).

Druhý počítač: CPU AMD Athlon 3000+ (64bit), operační systém Windows XP SP2 (32bit) a J2RE ve verzi *jre-1_5_0_02-windows-i586-p* (pro 32 bitové aplikace).

Program se dá spustit přímo kliknutím na soubor imagine.jar.[14] Nebo zadáním příkazu: *java –jar imagine.jar* (v příkazovém řádku).

6.1.2 Hlavní okno programu

Po spuštění se zobrazí hlavní okno programu, kde je v pravé části vnitřní okno pro výběr barvy a upřesnění vlastností konkrétních tvarů při kreslení. V horní části jsou možnosti pro otevření, či uložení souboru, ukončení programu, aplikaci filtrů na obraz a pro srovnání je použito i tří původních „java filtrů“.

6.1.3 Otevření souboru

Příkazem *File* a *Open* se otevře dialogové okno pro otevření souboru. Program, který je spuštěn pod JRE 1.5 podporuje formáty typu bmp, jpg a png. Tyto soubory se také zobrazí jako jediné možné, které lze otevřít.

6.1.4 Kreslení jednoduchých tvarů

Na obraz je možné malovat štětcem, kreslit čáry, obdélník a elipsu. Obrazce se při tahu myši vykreslují průběžně, podle aktuální pozice kurzoru, trvale jsou vykresleny až po

---

[14] Je potřebné mít asociovány soubory .JAR s *Java(TM) 2 Platform Standard Editon Library* aplikací. Po instalaci JRE je to už automaticky takto nastaveno.

uvolnění tlačítka myši. U těchto obrazců je možné zvolit tloušťku čáry a jejich barvu. Toto lze nastavit v pravé části obrazovky, v podokně *Properties*.

Při volbě „*Funny draw*“ se obrazce vykreslují trvale, tím mohou vzniknou zajímavé tvary.Viz obr.č. 18, kde byla modrou barvou kreslena elipsa, ale díky okamžitému vykreslování vznikl tvar kapky. U barev lze také nastavit hodnotu průhlednosti (alfa), nové tvary pak nepřekrývají původní obraz. Viz obr.č. 18, kde byl aplikován světle žlutý obdélník s nízkou hodnotou alfa.

6.1.5 Algoritmy pro úpravu obrazu

Na obraz lze aplikovat základní filtry, při jejich nastavení má uživatel možnost jedinečným způsobem zadat hodnoty pro provedení filtru. Viz kap. 6.2

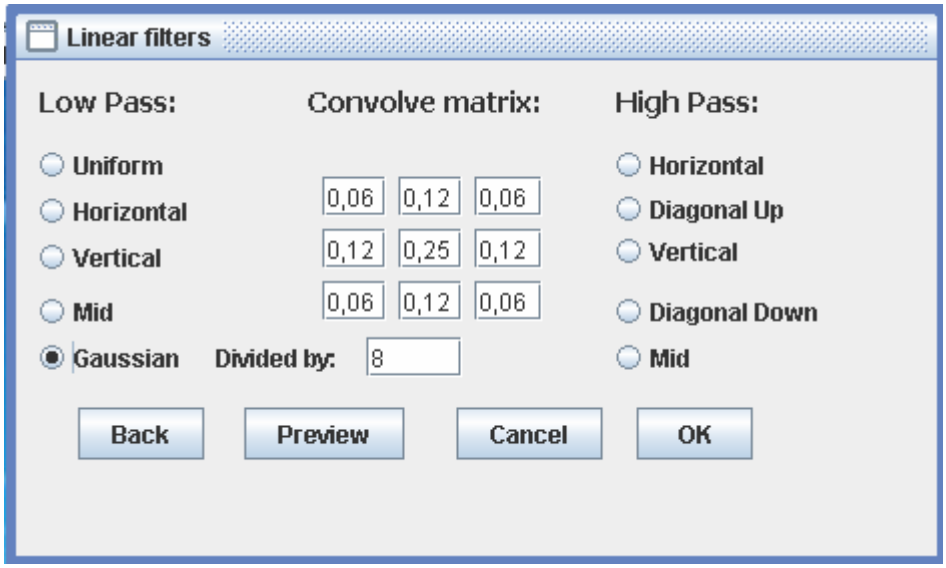
6.2 Práce s algoritmy

Následující popis se týká algoritmů, které byly speciálně vytvořeny pro tuto publikaci. Nabídka těchto metod se nachází pod volbou *"MyFilters"*. Podrobný popis těchto algoritmů je v kapitole 7.

6.2.1 Lineární filtry

Po zadání volby *„Linear Filters“* se zobrazí okno, viz obr. č. 19. V levé části má uživatel možnost zvolit mezi různými konvolučními maticemi pro filtraci vysokých frekvencí - *Low Pass* (obraz se rozmaže) a v pravé části lze vybírat mezi různými konvolučními maticemi pro filtraci nízkých frekvencí v obraze – *High Pass* (v obraze vyniknou hrany).

Při zvolení jednotlivých možností se nastavení konvoluční matice ukáže v prostřední části okna. Matice lze vždy libovolně upravit. Pomocí pole "Divided by" lze nastavit všechny čísla v matici naráz. Ukázka filtrace je v kap. 7.3.1

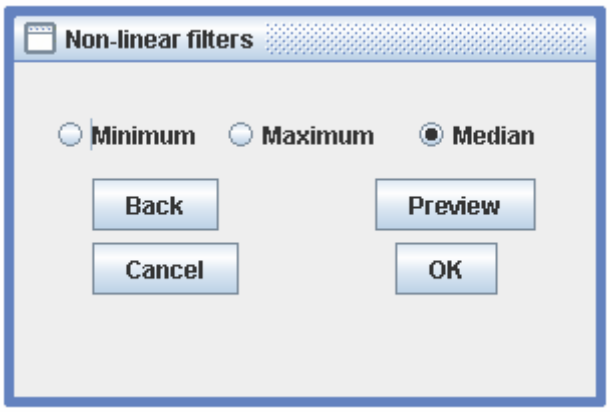


Obr. 19: Okno s nastavením pro aplikaci lineárních filtrů na obraz

Po stisknutí tlačítka „*Preview*" lze vyzkoušet efekt konkrétní matice, akci lze vzít zpět stisknutím tlačítka „*Back*". Po stisknutí tlačítka „OK“ se dané nastavení projeví na obraze a podokno „*Linear filters*“ se zavře.

6.2.2 Nelineární filtry

Po stisknutí položky v menu „*Non-linear Filters*“ se otevře okno s nastavením pro nelineární filtraci obrazu (obr. č.20), s možnostmi volby filtrace „*Minimum*“, „*Maximum*“, nebo „*Median*“.



Obr. 20: Okno s nastavením

pro nelineární filtraci obrazu

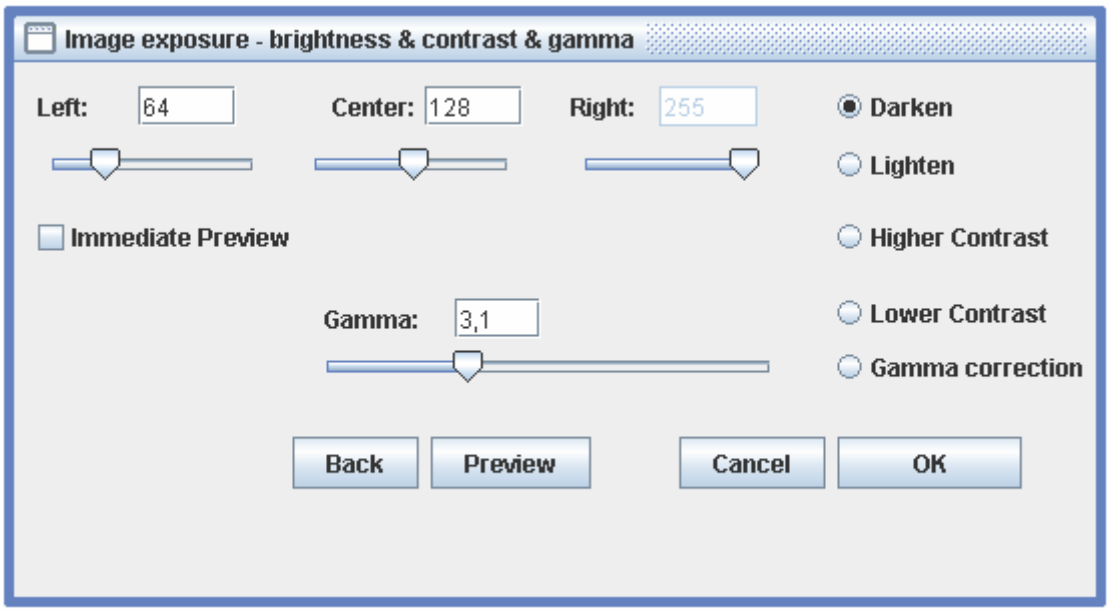
Po stisknutí tlačítka „*Preview*" lze vyzkoušet efekt konkrétní filtrace, akci lze vzít zpět stisknutím tlačítka „*Back*". Po stisknutí tlačítka „OK" se dané nastavení projeví na obraze a podokno „*Non-linear filters*" se zavře. Ukázka práce nelineárních filtrů je v kapitole 7.3.2.

6.2.3 Expozice obrazu

Tato funkce umožňuje měnit jas, kontrast a korekci gama v obraze. Po stisknutí volby „*Exposure*" se otevře okno (obr.č.21)

Na pravé straně lze volit mezi ztmavení- „*Darken*", zesvětlením – „*Lighten*", vyšším kontrastem – „*Higher Contrast*", nižším kontrastem „*Lower Contrast*" a změnou gama korekce „*Gamma correction*". Tyto algoritmy mají podobnou vlastnost jako známá a často používaná funkce, která se v mnoha programech jmenuje „***Křivky***" - *Curves*.

Při zatrženém tlačítku „*Immediate Preview*" lze sledovat okamžité změny v obraze podle aktuálního nastavení. Podrobný popis této funkce je v kapitole 7.2.2. a ukázka funkce toho algoritmu na obraze je v kapitole 7.3.3.



Obr. 21: Okno s nastaveními pro změnu jasu, kontrastu a gama korekce

- **Volba „Darken"** – lze měnit pozici posuvníku „*Left*" a posuvníku „*Center*". Tento algoritmus vytváří funkci, která změní RGB hodnoty pixelů podle hodnot „*Center*" a „*Left*" Čím větší bude rozdíl mezi těmito hodnotami, tím více se obraz ztmaví.

- **Volba „Lighten" –** lze měnit pozice posuvníků „*Center*" a „*Right*". Pro změnu hodnot pixelů s použije funkce, která vychází z těchto dvou hodnot. Čím větší bude rozdíl mezi těmito hodnotami, tím více se obraz zesvětlí. Jedná se o opak funkce „*Darken*"
- **Volba „Higher Contrast" –** zde lze měnit hodnoty „*Left*", „*Center*" a „*Right*". Čím větší bude rozdíl mezi hodnotami „*Left*" a „*Right*", tím větší kontrast vznikne. Hodnota „*Center*" určuje inflexní bod funkce
- **Volba „Lower Contrast" –** zde lze opět měnit všechny hodnoty, tj. „*Left*", „*Center*" i „*Right*". Čím větší bude rozdíl mezi hodnotami „*Left*" a „*Right*", tím více se kontrast v obraze sníží. Hodnota „*Center*" opět určuje inflexní bod funkce.
- **Volba „Gama correction"-** hodnota posuvníku „*Gamma*" určuje hodnotu gama korekce pro obraz v intervalu od 0 po 9.9.
  Doporučená hodnota je $\gamma = 2.5 \pm 0.3$ [1]

6.2.4 Negativ obrazu

Tato funkce invertuje barvy v obraze. Vytváří tak vždy opačné hodnoty barev. Ukázkový příklad této funkce na obraze je v kapitole 7.3.4.

6.2.5 Ekvalizace obrazu

Po stisknutí položky „*Equalization*" se otevře okno s volbou ekvalizace obrazu. Políčka „*Left Border*" a „*Right Border*" ukazují minimální, resp. maximální RGB hodnoty v obraze. Podle těchto hodnot se histogram obrazu poupraví („roztáhne") a tím se zlepší kvalita obrazu. Pokud je v polích hodnota 0 (pro „*Left Border*") a zároveň 255 (pro „*Right Border*"), tak histogram obrazu je v pořádku a obraz se nebude nijak měnit. Uživatel má možnost si libovolně zvolit hranice, tím ale dojde k degradaci barevné skladby obrazu. Ukázka této funkce na obraze je v kapitole 7.3.5.

6.3 Originální „java" filtry

V programu je také volba („*Orig Filters*"), která umožňuje aplikovat filtry Rozmazání („*Blur*"), Zaostření („*Sharp*") a také filtr Reliéf („*Relief*"). Tyto filtry nemají další podrobné nastavení a jsou v programu uvedeny pouze pro srovnání s ostatními

(vlastními) filtry. Hlavní rozdíl ve funkčnosti spočívá v tom, že filtrace vlastními algoritmy dokáže pracovat s obrázkem o maximální velikosti přibližně 1400x1050, doporučené maximální rozlišení je pak: 1200x900 pixelů. Kdežto originální "java" filtry je možné aplikovat i na mnohem větší obrazy. Vysvětlení problému - viz kapitola 5.6.1.

6.4 Architektura programu Imagine 1.0

Program je rozdělen na několik funkčních částí, které mohou být kdykoliv později libovolně doplňovány o další funkce. Struktura programu umožňuje případný další vývoj rozdělený i mezi více programátorů.

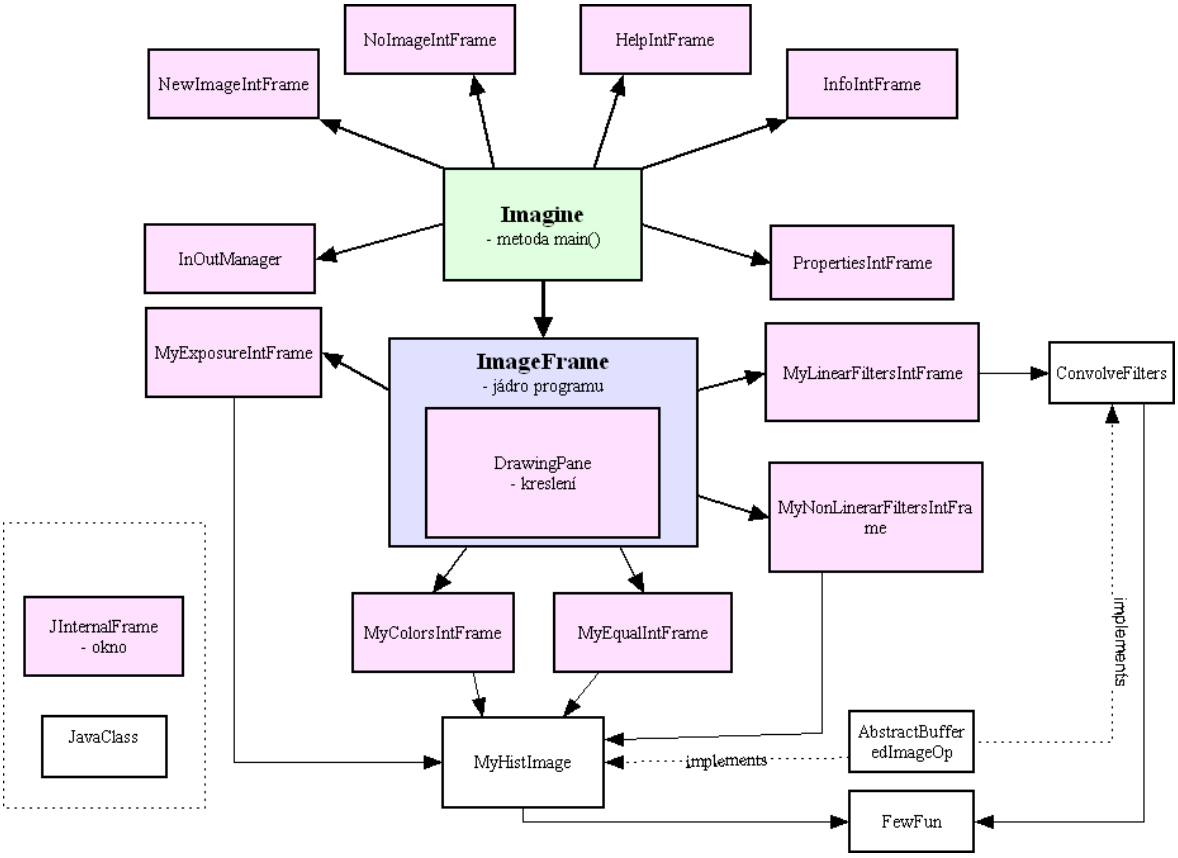
- Program se skládá ze 17 částí.

1. **Imagine.java –** obsahuje funkci *main()*, je tedy začátkem a také kostrou programu, zobrazuje základní okno programu (*JFrame*)
2. **PropertiesIntFrame.java** – její instance je spuštěna v konstruktoru Imagine.java, zobrazuje okno v pravé části aplikace, okno obsahuje komponenty pro nastavení různých vlastností.
3. **InOutManager.java**[15] **–** instance třídy spuštěna při otevírání, či zavírání souboru.
4. **NewImageIntFrame.java –** okno pro volbu rozměrů nového obrazu. Instance se vytváří v Imagine.java při volbě „*File - New*“.
5. **NoImageIntFrame.java –** okno, které upozorňuje, že není otevřen žádný soubor a vyzývá k otevření, či vytvoření obrazu.
6. **HelpIntFrame.java –** okno, s nápovědou k programu.
7. **InfoIntFrame.java –** vnitřní okno s informacemi o aplikaci.

---

[15] Převzato na základě volného copyrightu pro studijní účely a z části upraveno. © *2000-2003 San Diego Supercomputer Center (SDSC)*

8. **ImageFrame.java –** instance spuštěna při otevření, nebo vytvoření nového obrázku. Je jádrem programu, zobrazuje okno pro vykreslení obrázku. Vytváří instanci třídy *DrawingPane.java*, která vykresluje obraz.
9. **MyLinearFiltersIntFrame.java –** instance okna, která je spuštěna v Imagine.java volbou „*Linear Filters*“, zde lze volit mezi dolní, či horní propustí. Vytváří instanci *ConvolveFilters.java* a v ní spouští statickou funkci pro konvoluci.
10. **ConvolveFilters.java –** třída obsahující algoritmy lineárních filtrů. Dědí od třídy *AbstractBufferedImageOp.java*
11. **MyNonLinearFiltersIntFrame.java –** instance okna, která je spuštěna v *Imagine.java* volbou „*NonLinear Filters*“. Vytváří instanci *MyHistImage.java*.
12. **MyHistImage.java –** třída pro práci s histogramem. Obsahuje algoritmy pro úpravu jasu, kontrastu, gama korekce, barevného negativu a nelineárních filtrů.
13. **MyExposureIntFrame.java** – instance okna, která se spouští v *Imagine.java* (volba „*Exposure*"), ve kterém se volí nastavení pro úpravu jasu, kontrastu a gama korekci. Vytváří instanci třídy *MyHistImage.java*, která obsahuje algoritmy pro úpravy obrazu v histogramu.
14. **MyColorsIntFrame.java –** instance okna, která je spuštěna v *Imagine.java* volbou „*Negative*" ve kterém je volba pro negaci barev obrazu. Vytváří instanci třídy *MyHistImage.java*.
15. **MyEqualIntFrame.java** – instance okna, která je spuštěna v *Imagine.java* volbou „*Equalization*“, v tomto okně lze nastavit hranice histogramu pro ekvalizaci. Výchozí hodnoty jsou nastaveny na základě vyhodnocení obrazu. Vytváří instanci *MyHistImage.java*.
16. **AbstractBufferedImageOp.java –** třída pro správné získání RGB dat z objektu *BufferedImage*.
17. **FewFun.java –** třída obsahující drobné logické funkce pro práci s maticí.



Obr. 22: Logická struktura programu Imagine

6.4.1 Notace programového kódu

Názvy tříd začínají velkými písmeny a jejich instance začínají malými písmeny, viz např. *ImageFrame* (třída), *imageFrame* (instance). Statické finální proměnné jsou psány celé v kapitálkách (*ELLIPSE*). Nestatické proměnné začínají malým písmenem (*newImageWidth*). Názvy základních objektů (tlačítko, popisek) začínají velkým písmenem (*PreviewBT*) a jsou doplněny zkratkou své funkce – *BT* (*Button*), *LB* (*Label*), *RBT* (*RadioButton*). „*Frame*" na konci názvu třídy značí, že se jedná o objekt třídy *JFrame*, „*IntFrame*“ na konci názvu třídy pak znamená, že se jedná o objekt třídy *JInternalFrame*. „*Frame*" i „*InternalFrame*" jsou třídy, které se dají zobrazit jako okno. V názvech se nepoužívají podtržítka.

Vychází z maďarské notace[16]. Pokud se ji podaří striktně dodržovat, tak se program výrazně zpřehlední a sníží se pravděpodobnost vzniku chyby. Mezi další dobré „rady" patří psát před proměnnými jednoznakový prefix, který definuje o jaký typ proměnné se jedná (např. *s-string, i-int*). [16]

6.5 Imagine.java

Obsahuje funkci *main()*, je tedy začátkem programu.

V konstruktoru třídy se vytváří veškeré položky menu a registrují se jako posluchači událostí. Dále se vytváří nástrojová lišta s tlačítky. V třídě pak jsou naprogramovány reakce na události od stisknutí položek menu. Reakce jsou následující:

- **New** – posluchač volá funkci *MyNewPicture(...)*, která vytváří instanci třídy *NewImageIntFrame*
- **Open** – posluchač volá funkci *MyOpenPicture(...)*, která volá funkci instance *inOutManager*, ta vrací objekt typu File: *file = inOutManager.loadFile();* a následně vytváří instanci třídy ImageFrame, kde parametr konstruktoru uvádí objekt *file*.
- **Save** – volá funkci: *imageFrame.save WorkImg(Imagine JFrame);* Tato funkce, která je v třídě *ImageFrame*, pak volá funkci *save()* v *InOutManager*.
- **Exit** – testuje se, jestli existuje v paměti obraz a při úspěchu volá stejnou funkci pro uložení.
- **Linear Filters** – pokud existuje obraz v paměti, tak vytváří okno *MyLinearFiltersIntFrame*, jinak je spuštěno okno *NoImageIntFrame* (toto je testováno u všech filtrů).
- **NonLinearFilters** – vytváří okno *NonLinearFiltersIntFrame*
- **Exposure** – spouští okno *MyExposureIntFrame*
- **Negative** – spouští okno třídy *MyColorsIntFrame*

[16] podle legendárního programátora firmy Microsoft - Charlese Simonyi

- **Equalization** – spouští okno třídy *MyEqualIntFrame*
- **Help** – spouští instanci třídy *HelpIntFrame*
- **About** – spouští instanci třídy *AboutIntFrame*.

Veškerá interní okna (*JInternalFrame*) mají při vytváření instance jako parametr pro konstruktor objekt *ImagineJFrame*, ten zajišťuje vazbu na proměnné a funkce v celém programu. V jednotlivých třídách je objekt značen jako ***parent***.

6.6 PropertiesIntFrame.java

Interní okno (*JInternalFrame*), ve kterém lze nastavit vlastnosti pro kreslení.

Další funkcí je pak nastavení vlastností při kreslení, které se liší podle aktuálně vybraného tvaru. Ve spodní části okna se vykresluje aktuální pozice myši a velikost kresleného tvaru.

6.7 InOutManager

Třída převzata od San Diego Supercomputer Center, a to díky tomu, že pro studijní účely je možné tento kód používat a podle vlastní potřeby upravovat. Z velké části byl upraven pro účel této práce. Hlavní funkcí je načíst a uložit obraz, pro tento úkol se zobrazí dialogové okno se strukturou adresářů a také s možností zvolit příponu souboru. Parametrem pro uložení souboru je objekt typu *BufferedImage*, který předává obrazové informace. Objekt *File* je zjištěn uživatelem - z adresářové struktury a zadáním názvu a přípony souboru.

6.8 ImageFrame – hlavní část programu

Třída je typu *JFrame* a v jejím oknu se zobrazuje veškerý průběh práce. Obsahuje vnitřní třídu *DrawingPane*, která má zaregistrovány posluchače na události od myši.

6.8.1 Funkce paintCompoment()

V třídě je překryta funkce *paintComponent(Graphics g)*, ta se spouští vždy, když vznikne požadavek na překreslení okna. Požadavek na překreslení vzniká vždy, když se okna dotkne kursor myši, nebo pokud je událost vyvolána příkazem *repaint()*. V této funkci se nejdříve testuje, zdali je načten obrázek *(if (workImg == null)*, pokud ne, tak se do objektu typu *BufferedImage* načte nový obrázek: *originImg = ImageIO.read(file);*

Následně je vytvořen druhý objekt typu *BufferedImage* a to se stejnými rozměry jako má *originImg*: *workImg = (BufferedImage)this.createImage(workImgW, workImgH);* Podle *workImg* je vytvořen grafický kontext:

Graphics2D gc = workImg.createGraphics(); a do toho grafického kontextu je vykreslen *originImg*: *gc.drawImage(originImg,null, 0,0);*

- Ve funkci *paintComponent(Graphics g)* je pak při každém dalším požadavku na kreslení opakována následující část:

Do grafického kontextu je překreslen aktuální obraz: *g2.drawImage(workImg, null, 0, 0);* V případě, že byl obraz přiblížen, nebo vzdálen funkcí zoom, tak se do grafického kontextu vykreslí *zoomImg*, což je opět objekt typu *BufferedImage*, ale upravený affiní transformací.

Následně je volána funkce *drawShape*, ta na základě proměnné *shapeType*, vykresluje konkrétní tvar s aktuálním nastavením pozice, tvaru a barvy, viz např. obdélník: *g2.fill(new Rectangle2D.Double(XPos, YPos, XSize, YSize));*

6.8.2 Události od myši

- **mouseReleased(MouseEvent e)** - uvolnění tlačítka myši, jsou aktualizovány proměnné: *XEnd = e.getX(); YEnd = e.getY();* V případě změněné velikosti obrazu je proveden přepočet polohy na původní obraz *workImg* – ten se při každém požadavku na překreslení přepočítá na velikost *zoomImg*. Dále je volána funkce *drawShape()* pro vykreslení nového tvaru. Na závěr je spuštěn požadavek na překreslení okna: *this.repaint();*
- **mousePressed(MouseEvent e)** – stlačením tlačítka myši, jsou aktualizovány proměnné: *XStart = e.getX(); YStart = e.getY();*

- **mouseDragged(MouseEvent e)** - pohyb myši při stisknutém tlačítku. Jsou aktualizovány proměnné: *XPos = e.getX(); YPos = e.getY();* Následuje poměrně složitý výpočet pro vykreslení tvarů[17].
- **mouseMoved(MouseEvent e)** – pohyb myši. Je aktualizováno číslo v *PropertiesIntFrame*, které udává pozici kursoru myši.

[17] Je třeba vzít v úvahu, že pokud uživatel kreslí např. obdélník, tak během tahu se může vrátit i přes pozici, kde se tvar začínal vykreslovat a konečná pozice obdélníku může být menší než počáteční. Toto algoritmus bere na vědomí.

7 VLASTNÍ ALGORITMY

V programu jsou použity vlastně vytvořené algoritmy pro úpravu obrazu. V této kapitole budou popsány a to i jak byly odvozeny. Tyto filtry vždy nejdříve získají obrazová data z obrazu a uloží je jako vektor typu int. Číselné hodnoty reprezentují ARGB (alfa, červená, zelená a modrá) hodnoty. Zde u funkce *getRGB* dochází k určitému omezení velikosti zpracovávaného souboru. Funkce *getRGB* totiž vrací vektor *int[] inPixels = new int[width*height];* ve kterém jsou uloženy všechny RGB hodnoty obrazu. Z něj se bitovým posunem získají konkrétní r,g,b hodnoty. Rozsah *int* je pak už malý pro obrazy od velikosti okolo 1,5 MPix (např. 1400 na 1050 pixelů). Viz kap. 5.6.1.

7.1 ConvolveFilters – lineární filtry

Třída dědí od *AbstractBufferedImageOp*, parametrem konstruktoru je

float[] FilterMatrix, který udává jaká konvoluční matice se bude používat. Důležitou funkcí je *getRGB(src, 0, 0, width, height, inPixels);* která v parametru *src* vrací objekt typu *BufferedImage*. Hlavní funkcí je *convolveHV()*, která obsahuje samotný konvoluční algoritmus. Veškerá nastavení pro tento algoritmus vyplývají z konvoluční matice, která se definuje v okně *MyLinearFiltersIntFrame*. Algoritmus cyklem „*for*“ prochází celý vektor int, který obsahuje obrazová data a provádí konvoluci, viz také kapitola 4. Ukázka práce lineárních filtrů je v kapitole 7.3.1.

7.2 MyHistImage – expozice obrazu a nelineární filtry

Třída pro úpravu obrazu pomocí informací z histogramu. Parametr konstruktoru je objekt typu *BufferedImage*. Aby bylo možné používat funkci *getRGB*, tak třída dědí od *AbstractBufferedImageOp*.

7.2.1 Funkce passImg()

Funkce *passImg(...)* má za úkol projít obrazová data a zjistit potřebné informace o obraze. Tato funkce nahrazuje do jisté míry funkci histogramu.

Důležitá část je, kdy se pravidelně ve dvou cyklech *for* prochází vzestupně řádky a sloupce a získávají se data obrazu:

```
int rgb = inPixels[y*width+x];
```

```
a = ((rgb >> 24) & 0xff);
r = ((rgb >> 16) & 0xff);
g = ((rgb >> 8) & 0xff);
b = (rgb & 0xff);
```

bitovým posunem jsou z vektoru typu *int[]* získány konkrétní r,g,b obrazová data uložená v samostatných proměnných.

Podle vzorce pro výpočet jasu z hodnot RGB je zjištěna proměnná brightness [1]:

*brightness[index]=(int)(0.299*FewFun.clamp(r+0.5) + 0.587*FewFun.clamp(g+0.5) + 0.114*FewFun.clamp(b+0.5));*[18]

7.2.2 Funkce exposureImg()

Jeden z parametrů funkce je proměnná *int type*, ta udává co bude funkce vykonávat. 1 - ztmavení, 2 - zesvětlení, 3 - vyšší kontrast, 4-nižší kontrast. 5- gama korekce. Ukázka těchto algoritmů je v kapitole 7.3.3.

Ve vnitřním okně uživatel zadá hodnotu „*left*“, „*center*“ a „*right*“, tyto hodnoty se použijí pro výpočet funkce. Veškeré RGB hodnoty se pomocí této funkce přepočítají na hodnoty nové. Úpravu histogramu by bylo vhodné provádět v jiném barevném prostoru než je RGB (například YUV), tento algoritmus mění hodnoty všech tří složek (*ir, ig, ib*) stejným poměrem, tím pak dochází k barevnému rozptylu[19]. Vhodnější pro tuto funkci jsou barevné prostory, které mají jasovou složku definovanou odděleně od barev.

- **Ztmavení –** zde jsou zadány hodnoty *„left*“ a „*center*“. To znamená, že pixely s RGB hodnotou „*center*“ budou mít ve výsledku hodnotu „*left*“ (hodnota se sníží). Viz obrázek.č.23.

[18] Funkce FewFun.clamp zajišťuje, aby se hodnoty nedostaly mimo mez <0;255>

[19] Lze si všimnout, pokud uživatel v okně pro ekvalizaci zadá několikrát za sebou „Preview“ a „Back“., nebo když v okně pro změnu kontrastu zadá krajní hodnoty.

Funkce pro snížení jasu je:

$$i' = \frac{i^{center/left}}{\max^{(center/left)-1}}$$

Kde *i* jsou R,G,B hodnoty, *max* je maximální hodnota - zde 255.

Pro výpočet funkce se musí nejdříve spočítat hodnota *dx = (double)center/(double)left;*

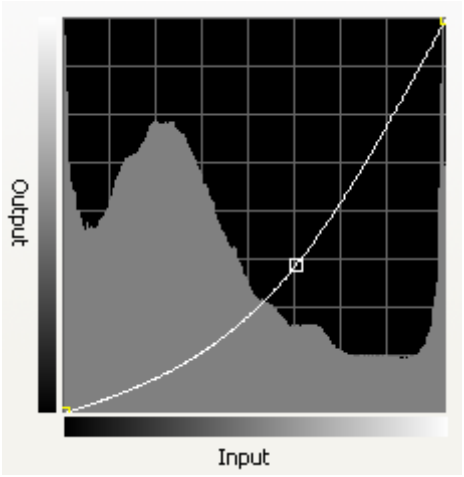
hodnota *max = Math.pow(max, dx-one);*

a poté je možné vypočítat nové RGB hodnoty:

ir[index]=(int)((Math.pow((double)ir[index], dx))/max);

ig[index]=(int)((Math.pow((double)ig[index], dx))/max);

ib[index]=(int)((Math.pow((double)ib[index], dx))/max);



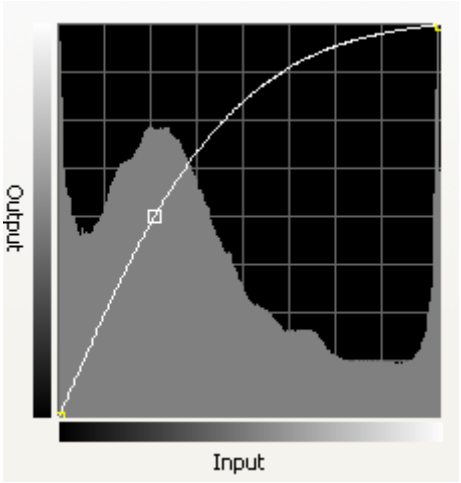
Obr. 23: Ukázka průběhu funkce pro ztmavení

- **Zesvětlení –** uživatel zde zadá hodnoty *right* a *center*. Zde to znamená, že RGB hodnoty u pixelů s hodnotou *center* se přepočítají na hodnotu *right*

  (hodnota se zvýší), viz obr. č.24. Všechny pixely budou přepočítány podle vztahu:

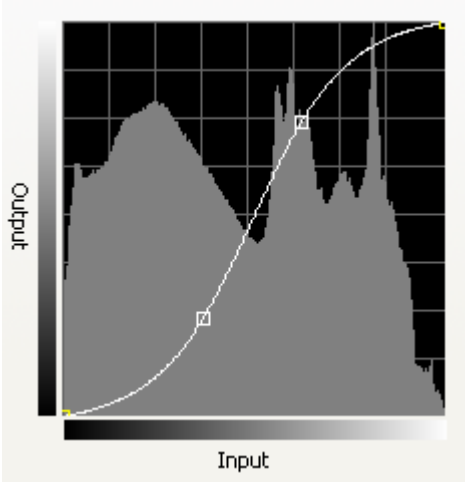
$$i' = \max^{\frac{right}{center}-1} \cdot \sqrt[center]{i^{right}}$$

```
dx = (double)right/(double)center;
max = Math.pow(max,dx-one);
ir[index]=(int)Math.pow(((double)ir[index])*max, 1/dx);
ig[index]=(int)Math.pow(((double)ig[index])*max, 1/dx);
ib[index]=(int)Math.pow(((double)ib[index])*max, 1/dx);
```



Obr. 24: Ukázka funkce pro zesvětlení obrazu

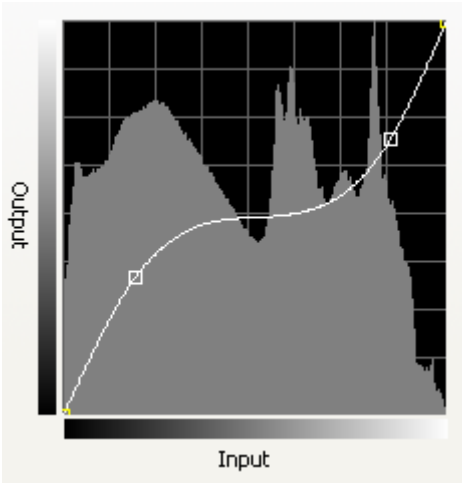
- **Vyšší kontrast –** obsahuje funkci, která je vlastně spojením obou předchozích. Uživatel zadává tři hodnoty, *left, center* a *right*. Je potřeba vytvořit takovou funkci, jak je uvedeno na obrázku č. 25.



Obr. 25: Funkce pro zvýšení kontrastu

Tím lze dojít k závěru, že první část křivky je shodná s funkcí pro ztmavení a druhá část křivky s funkcí pro zesvětlení. To znamená, že na interval hodnot 0 až *center* se aplikuje vzorec pro ztmavení a na interval *center* až 255 se aplikuje vzorec pro zesvětlení obrazu.

- **Snížení kontrastu obrazu –** je přesným opakem předchozí funkce. To znamená, že pro interval 0 až *center* se aplikuje vzorec pro zesvětlení a pro interval *center* až 255 se aplikuje vzorec pro ztmavení obrazu, funkci ukazuje obrázek č.26.



Obr. 26: Funkce pro snížení kontrastu v obraze

- **Gama korekce –** RGB hodnoty pixelů jsou přepočítány podle vztahu:

$$i' = i^{\frac{1}{\gamma}}$$

  Při výpočtu je pouze nutné, aby se hodnoty z intervalu 0 - 255 nejdříve přepočítaly na interval 0 až 1.

- **Ekvalizace histogramu –** v konstruktoru třídy *MyEqualIntFrame* je volána funkce pro vypočet nejnižší a nejvyšší hodnoty. Tyto čísla jsou pak vypsána ve vnitřním okně. Uživatel je může pozměnit a nebo ponechat. Algoritmus pak pomocí funkce přepočítá hodnoty RGB u pixelů tak, že minimální hodnota bude 0 a maximální hodnota RGB bude 255. Například pro červenou barvu je programový kód následující:

  dir=(double)(ir[index])/max;

temp = Math.pow(dir, one/gamma);

*ir[index]=(int)(temp*max);*

Ukázka ekvalizace na obraze je v kapitole 7.3.5.

- **Negativ obrazu -** Funkce *changeColor()*

  Má za úkol vytvořit negativní obraz. K tomu postačí pouze následující:
  ir[index]= 255-ir[index];

  ig[index]= 255-ig[index];

  ib[index]= 255-ib[index];

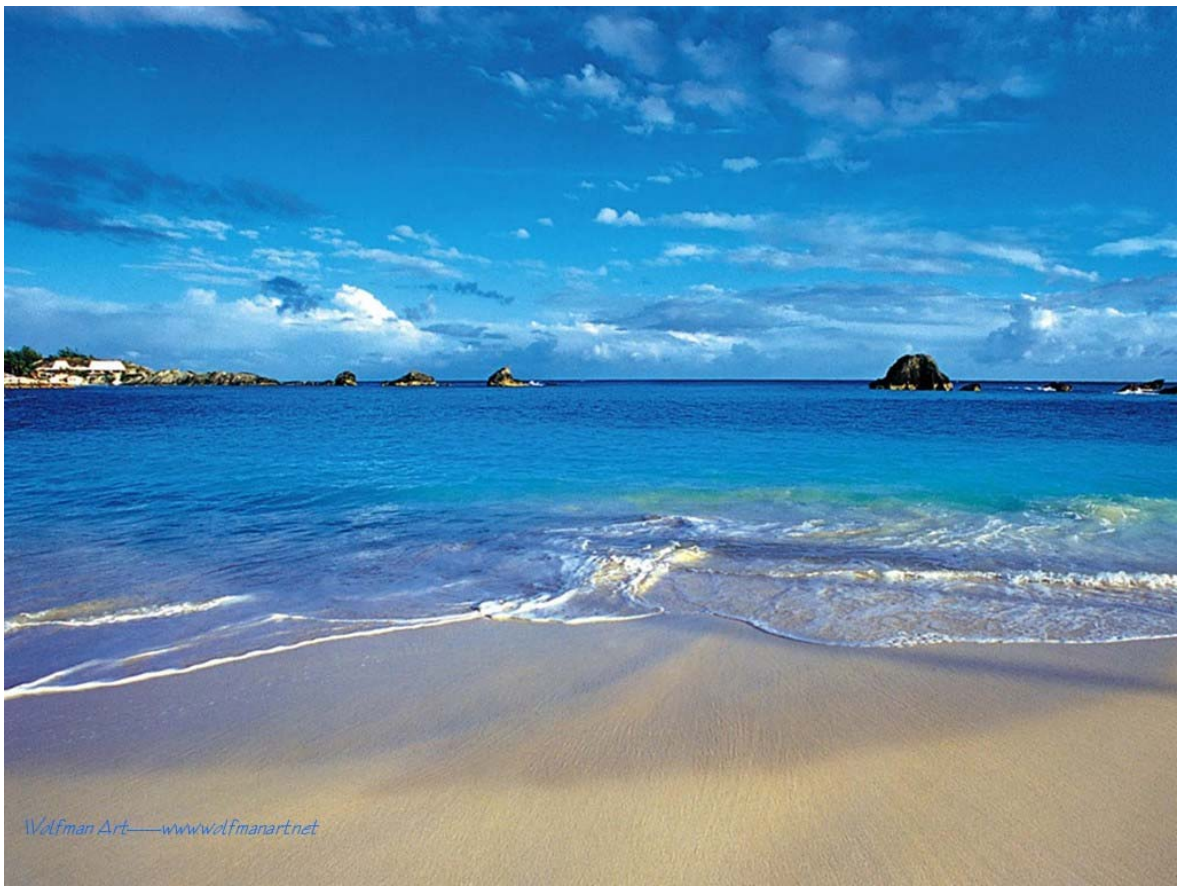
  Neboli k hodnotám barev se vytvoří doplňky do hodnoty 255. Ukázka negativu obrazu je v kapitole 7.3.4.

7.2.3 Funkce NonLinearFilters() – nelineární filtry

Funkce vrací objekt typu *BufferedImage* a má za úkol aplikovat nelineární filtry typu *minimum*, *medián* a *maximum*. U filtraci typu *minimum* a *maximum* se u každého pixelu v obraze prohledá jeho okolí (podle matice 3x3) a na daný pixel aplikuje minimální, resp. maximální hodnotu z daného okolí. U filtru typu *medián* je aplikována hodnota, která je v okolí 3x3 nejčastější.

7.3 Ukázkové aplikace vlastních algoritmů na obraz

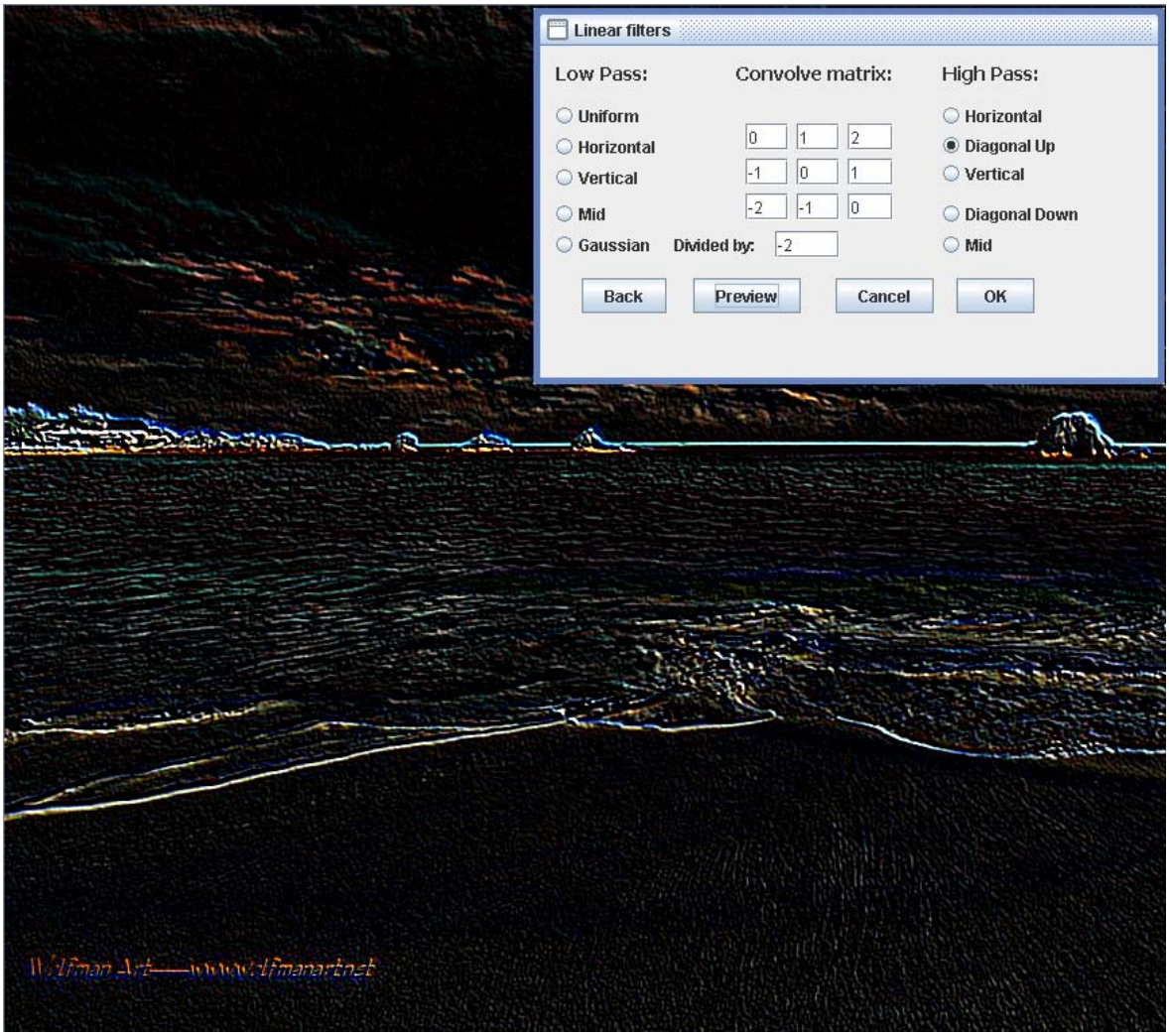
Tato kapitola obsahuje ukázky práce vlastních algoritmů. Jedná se o zachycení výsledků naprogramovaných filtrů v programu Imagine 1.0. Ke každému obrazu je přidáno i okno s nastavením, které se použilo při aplikaci algoritmu na obraz.



Obr. 27: Originální obraz, bez filtrací.

7.3.1 Lineární filtry

Na obraz byl aplikován lineární filtr s horní propustí, je zde patrné silné zvýraznění hran (přechod moře a oblohy). Nastavení je uvedeno v podokně.

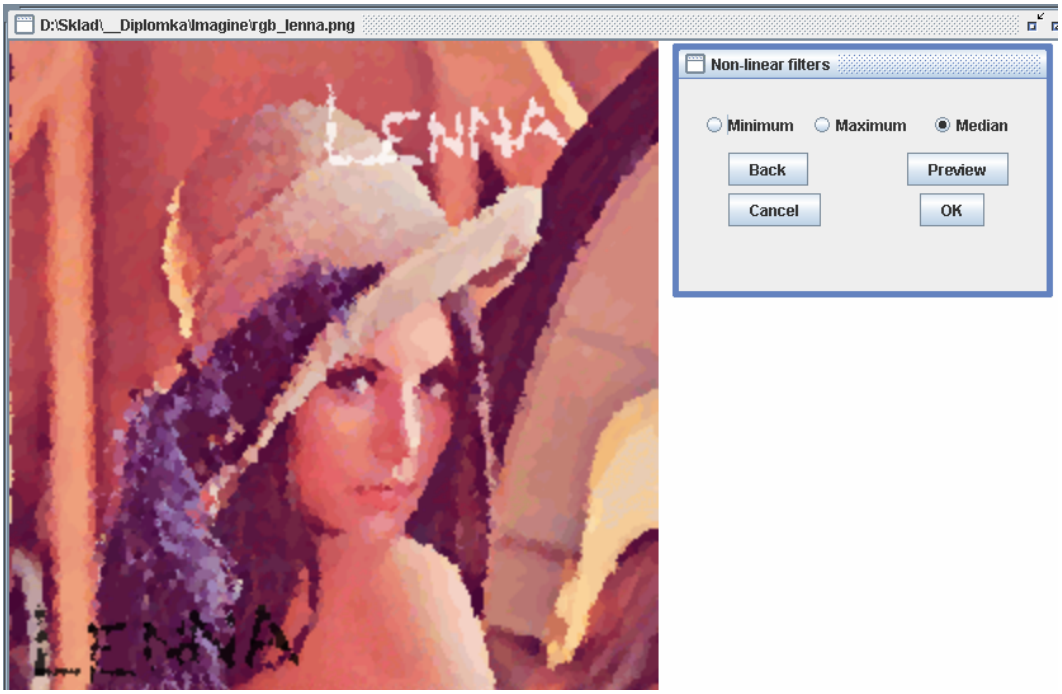


Obr. 28: Obraz po aplikaci lineárním filtrem typu horní propust.

7.3.2 Nelineární filtry

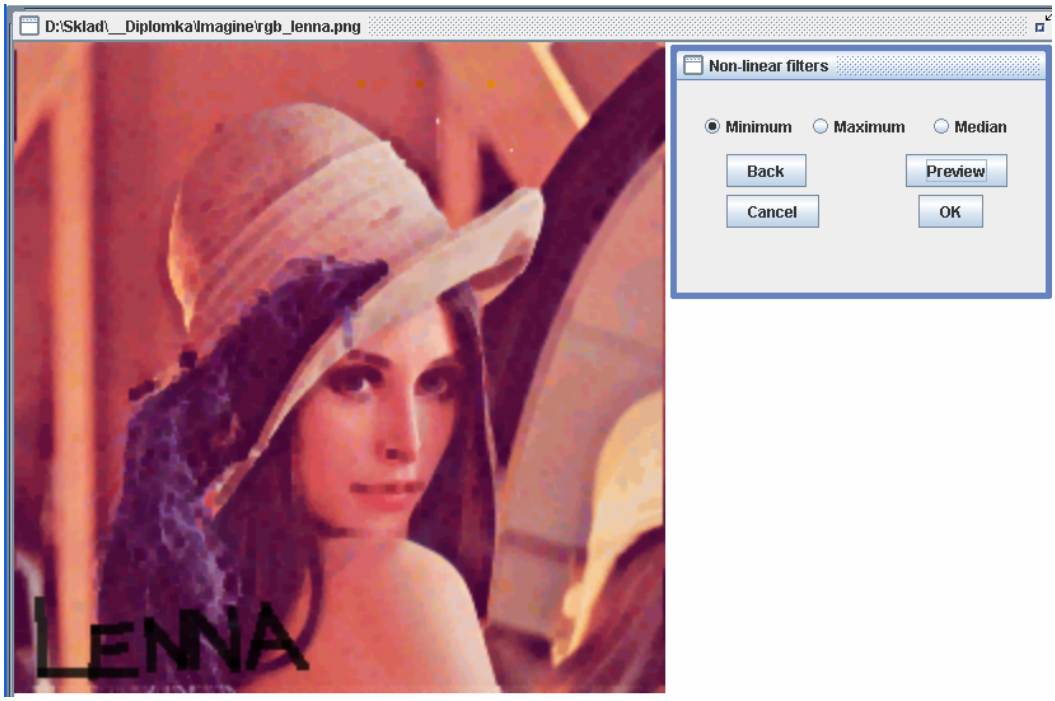
Obr. 29: Vzorový obraz „Lenna" pro nelineární filtraci

- Na obraz byl třikrát po sobě aplikován filtr „*median*", vytvořil se tak výrazný umělecký efekt štětce.



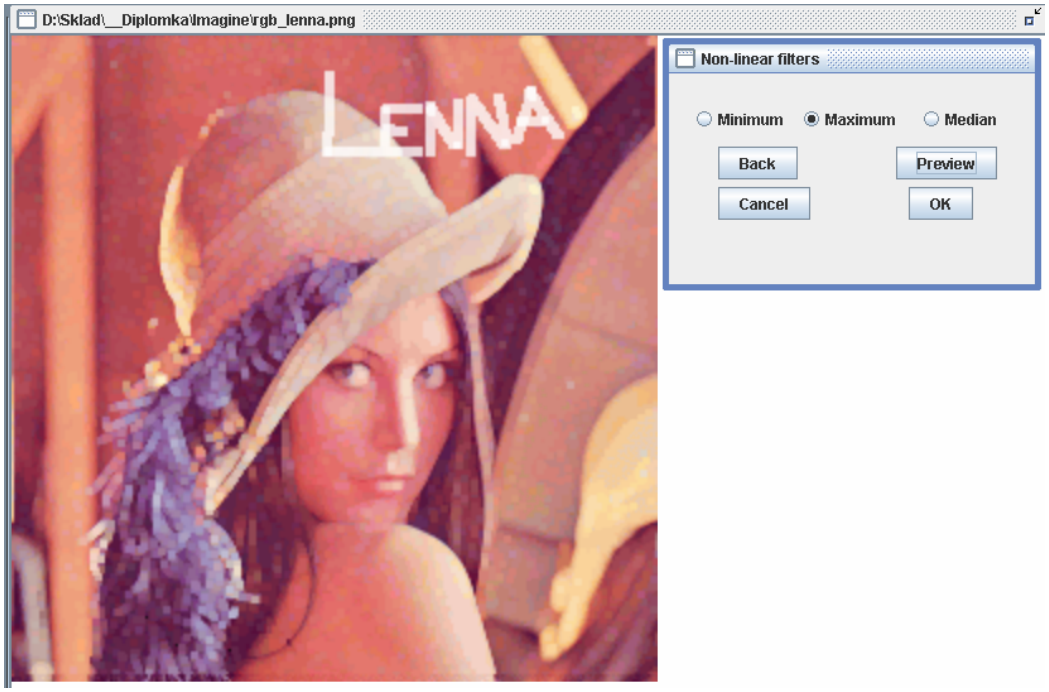
Obr. 30: „Lenna“ po aplikaci filtru „median“

- Na obraz aplikován nelineární filtr typu „*minimum*“, je zde patrné zvětšení tmavých částí a úplné odfiltrování bílého nápisu „Lenna".



Obr. 31: Ztráta bílých čar po filtraci typu „minimum“

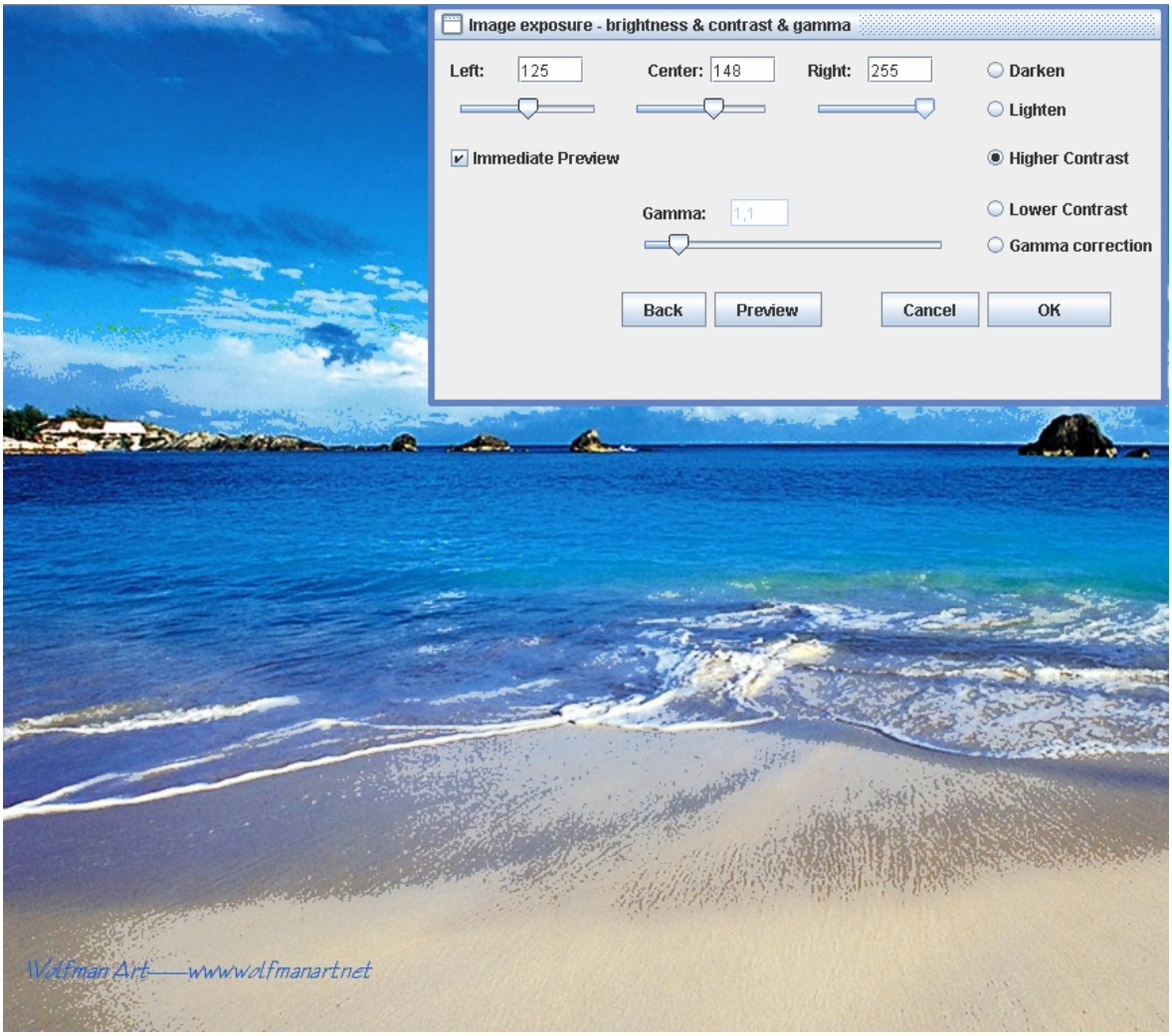
- Opačný efekt má filtr typu „maximum“, tedy zvětšení světlých čar a filtraci tmavých čar. Zde se zcela odfiltroval černý nápis „Lenna“.



Obr. 32: Po aplikaci nelineárním filtrem typu „maximum“

7.3.3 Expozice obrazu – křivky

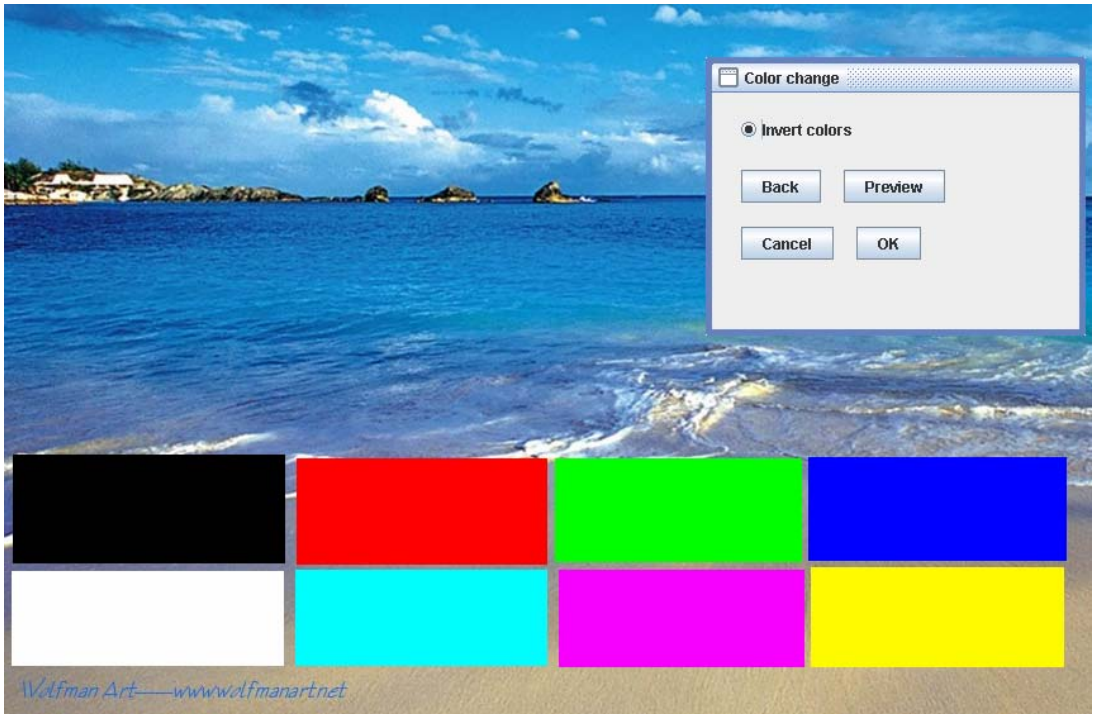
Obrazu byl dodán kontrast. Z nastavení lze vyčíst, že kontrast byl zvýšen ve světlejších částech obrazu. Zvýšení kontrastu je patrné hlavně v oblasti pláže.



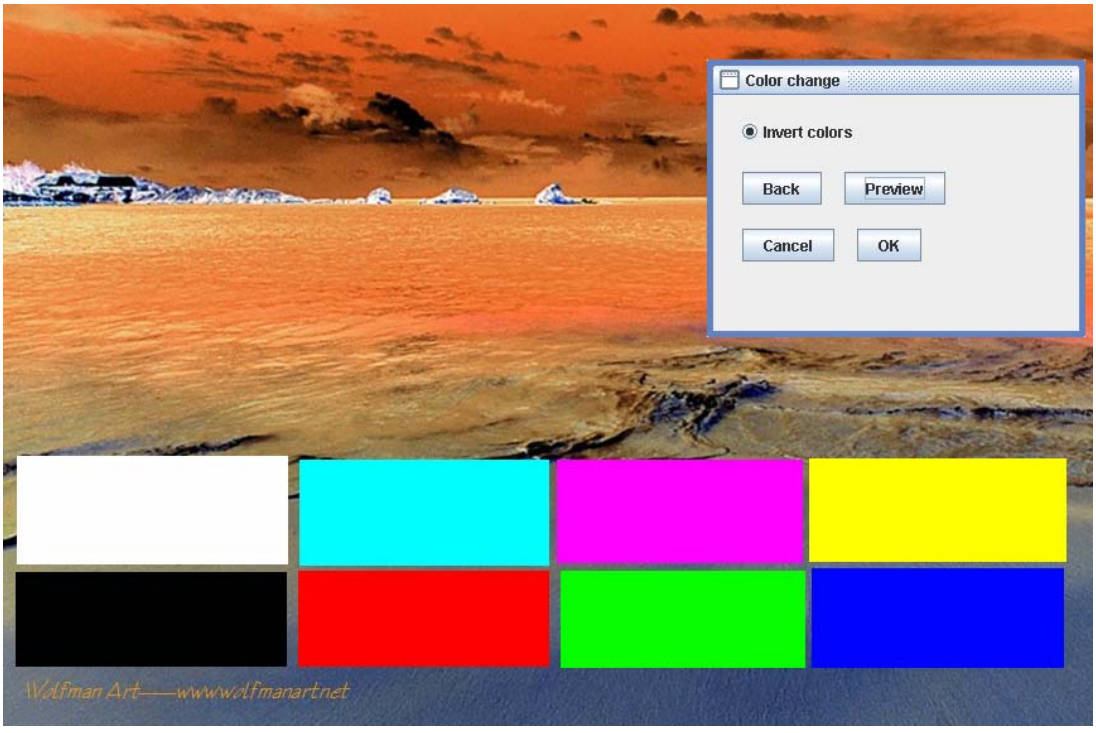
Obr. 33: Po zvýšení kontrastu

7.3.4 Negace

Byl vytvořen negativ obrazu. Zde je možné sledovat změnu barev podle barevných polí ve spodní části obrazu.(„*red->cyan, green->magenta, blue->yellow*“)



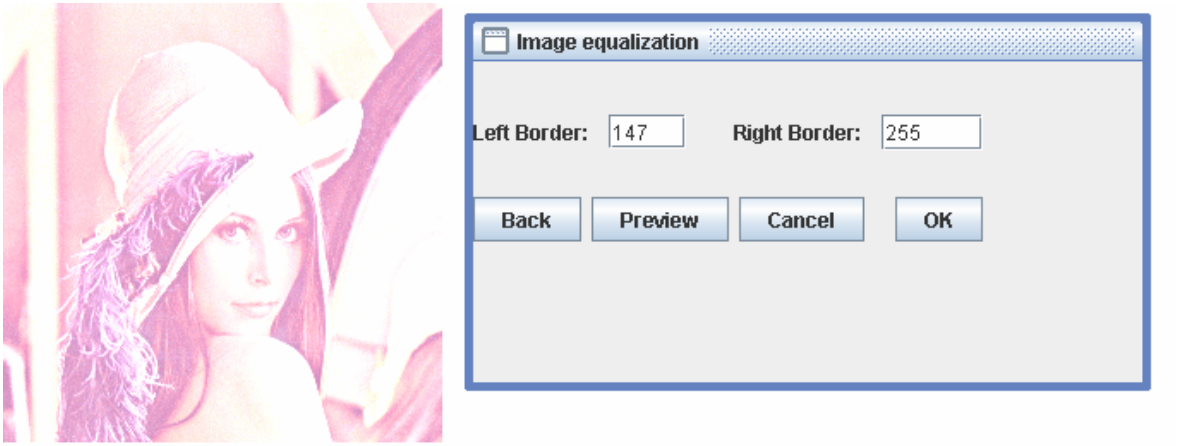
Obr. 34: Původní obraz s barevnými poli RGB a CMY



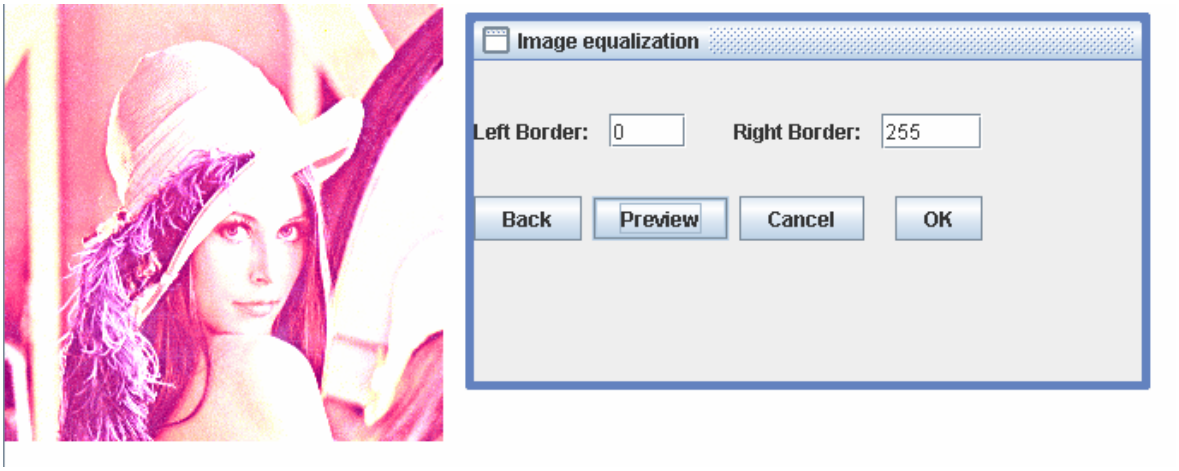
Obr. 35: Negativ obrazu s barevnými poli CMY a RGB

7.3.5 Ekvalizace

Obr. č.36 ukazuje obrázek s úzkým histogramem, tj. obraz, který nevyužívá všech dostupných intenzit. Vnitřní okno ukazuje zjištěné minimální a maximální hodnoty v histogramu. Na Obr. č. 37 je pak upravený obraz po ekvalizaci.



Obr. 36: Před ekvalizací



Obr. 37: Po ekvalizaci

ZÁVĚR

Program Imagine je zcela samostatný grafických editor pro zpracování bitmapových souborů, který obsahuje několik propracovaných funkcí. Skoro při každém spuštění programu *Imagine* mne napadali další a další možnosti, které by bylo dobré do programu implementovat. Při grafické tvorbě se naplno uplatňuje lidská tvořivost a díky tomu lze téměř neomezeně vymýšlet nové funkce pro grafické úpravy. Nejtěžší částí bylo vytvořit stabilní jádro programu tak, aby dokázalo obsluhovat různé požadavky uživatele. Tento úkol se mi podařil a poté mi už nic nebránilo popustit uzdu své fantazii a tvořivosti a vymyslet nové funkce programu.

Program *Imagine* tedy dokáže následující:

- Zápis, čtení a vytváření grafických souborů typu BMP, JPG a PNG.
- Malování štětcem a kreslení různých tvarů.
- Rozmazání obrazu v různých směrech (*„Linear Filters“* → *„Low Pass“*)
- Zvýraznění hran v obraze (*„Linear Filters“* → *„High Pass“*)
- Vytvoření plastického obrazu (*„Linear Filters“* → *„High Pass“* → *„Negative“*)
- Filtrace tmavých čar („*Non-linear Filters“* → *„maximum“*)
- Filtrace světlých čar a zvýraznění čar tmavých („*Non-linear Filters“* → *„minimum“*)
- Vytvoření efektu „štětec“ („*Non-linear Filters“* → *„median“*)
- Změna jasu, kontrastu a gama korekce(„*Exposure“*), toto je podobné funkci, v mnoha programech nazvané jako „křivky“.
- Vytvoření negativu obrazu („*Negative“*)
- Ekvalizace histogramu („*Equalization“*)

Při používání těchto funkcí je uživateli alespoň z části umožněno nahlédnout do způsobu práce algoritmů. Například poněkud netradiční je, že si uživatel může sám vytvořit vlastní konvoluční matici pro lineární filtraci.

Při tvorbě filtrací jsem nepoužíval už připravené metody, ale vytvořil jsem vlastní algoritmy, které pracují s každým obrazovým bodem zvlášť. Tento postup dává velkou

volnost programátorovi, který pak může téměř neomezeně vytvářet nové úpravy. Mezi omezení, která přináší tento postup, patří maximální rozměry obrazu (asi 1500 na 1200 pixelů). Rychlost práce nových algoritmů je jen o něco málo pomalejší, než u připravených metod.

Nespornou výhodou použití Javy, je možnost program spustit na kterémkoliv operačním systému. Zdrojový kód je transparentní a byla k němu vytvořena anglická dokumentace v html souborech. Případný zájemce o programování v tomto jazyce má tedy možnost najít v této práci inspiraci pro své programy a s trochou vynalézavosti vymyslet vlastní postupy.

Díky své segmentaci Java umožňuje tvorbu aplikací pro servery, osobní počítače, ale i pro mobilní telefony. Například JME je verzí Javy pro mobilní telefony. Její programové konstrukce se jen z části liší od standardní edice (JSE). Po překonání několika technických problémů je pak možné využít kódu pro standardní edici Javy a vytvořit z něj aplikaci pro mobilní telefon.

Java je univerzální, přívětivý a poměrně výkonný programovací jazyk a jako takový skrývá velký potenciál k rozvoji v mnoha oblastech, včetně počítačové grafiky.

SEZNAM POUŽITÉ LITERATURY

[1] ŽÁRA, J., BENEŠ, B., SOCHOR, J., FELKEL, P.: *Moderní počítačová grafika*, Computerpress, Brno, 2004. ISBN 80-251-0454-0

[2] HEROUT, P.: *Učebnice jazyka Java*, Kopp, České Budějovice, 2000. ISBN 80-7232-115-3

[3] HEROUT, P.: *Java – grafické uživatelské prostředí a čeština,* Kopp, České Budějovice, 2004. ISBN 80-7232-237-0

[4] HEROUT, P.: *Java – bohatství knihoven*, Kopp, České Budějovice, 2003. ISBN 80-7232-209-5

[5] MARTIŠEK, D.: *Matematické principy grafických systémů*, Littera, Brno, 2002. ISBN 80-85763-19-2

[6] HAWLITZEK, F.: *Java2 – příručka programátora*, Grada, Praha, 2002. ISBN 80-247-9060-2

[7] KISZKA, B.: *1001 tipů a triků pro programování v jazyce Java*, Computerpress, Brno, 2003.ISBN 80-7226-989-5

[8] ECKEL, B.: *Myslíme v jazyku Java – knihovna zkušeného programátora*, Grada, Praha, 2001. ISBN 80-247-0027-1

[9] PELIKÁN, J., SOCHOR, J.: *Barva a barevné vidění,* [online]. [cit. 2006-05-07]. Dostupný z WWW: <www.fi.muni.cz/~sochor/M4730/Slajdy/Barvy.pdf>

[10]*Wikipedia encyklopedie* [online]. [cit. 2006-05-08]. Dostupný z WWW: <http://cs.wikipedia.org/wiki/Hlavní_strana>.

[11] HAINA, P.: *Programování v jazyku Java (1) – Úvod* [online] 9.7.2004 [cit. 2006-05-20]. Dostupný z WWW: <http://www.linuxsoft.cz/article.php?id_article=244>.

[12] KOUBA, T.: *Porovnání rychlosti Javy na MS Windows a Linuxu a srovnání s C# a C* [online] 17.10.2003 [cit. 2006-04-20]. Dostupný z WWW: <www.neo.cz/~tomas/java.net/2003-10.html>.

[13] PAVELEK, M. JANOTKOVÁ, E..: *Vizualizační a optické měřicí metody,* FS VUT, Brno, 2001 [online] prosinec 2001 [cit. 2006-04-25]. Dostupný z WWW: <http://dt.fme.vutbr.cz/~pavelek/optika>

[14] JELÍNEK, L.: *Java (13) - JDK, vývojová prostředí* [online] 5.5.2005[cit.12.5.2006]. Dostupný z WWW: http://www.linuxsoft.cz/article.php?id_article=797>

[15] ALIMANT, F.: *JDK 5.0 Documentation* [online], 10.10.2004 [cit. 2006-03-17]. Dostupný z WWW: <http://www.allimant.org/javadoc/indexe.html>

[16] LEDERBUCH, P.: *Rozbor prvního programu, konvence při programování* [online], 26.11.2001 [cit. 2006-05-07].
Dostupný z WWW: <http://iason.zcu.cz/~lederbuc/local/win/prednes3.html>

SEZNAM OBRÁZKŮ

Obr. 1: Barevný rozsah 16-ti barev 11
Obr. 2: Aditivní způsob skládání barev u RGB 12
Obr. 3: Rozdíly ve vyjádření modelů RGB(vlevo) a CMY(vpravo) 13
Obr. 4: Subtraktivní způsob 14
skládání barev u modelu CMYK 14
Obr. 5: Zvětšení - vlevo u rastrového 17
a vpravo u vektorového obrázku 17
Obr. 6: Otočení obrazu s horizontální osnovou proužků o malý úhel 23
Obr. 7: Moaré obrazce při 23
nedostatečném počtu obrazových bodů 23
Obr. 8: Obraz a vazby na histogram 26
Obr. 9: Histogramy, vlevo tmavý obraz - low-key, ... 27
vpravo pak jasný obraz - high-key 27
Obr. 10: Vlevo je histogram neostrého obrazu - mid-key, vpravo pak histogram obrazu s vysokým kontrastem. 27
Obr. 11: Histogram obrazu, který 28
nevyužívá všechny úrovně intenzit 28
Obr. 12: Histogram obrazu 29
po ekvalizaci 29
Obr. 13: Grafické znázornění 30
funkce kontrast a jas 30
Obr. 14: Grafické znázornění 30
funkce negativ 30
Obr. 15: Příklady různých typů 32
symetrických okolí bodu 32
Obr. 16: Příklady matic váhových koeficientů 33
pro filtry typu dolní propust 34
Obr. 17: Příklad matice váhových koeficientů 34
pro filtr typu horní propust 34

Obr. 18: Program Imagine a ukázka jeho funkcí..... 46

Obr. 19: Okno s nastavením pro aplikaci lineárních filtrů na obraz 49

Obr. 20: Okno s nastavením pro nelineární filtraci obrazu..... 49

Obr. 21: Okno s nastaveními pro změnu jasu, kontrastu a gama korekce 50

Obr. 22: Logická struktura programu Imagine 54

Obr. 23: Ukázka průběhu funkce pro ztmavení 63

Obr. 24: Ukázka funkce pro zesvětlení obrazu.....64

Obr. 25: Funkce pro zvýšení kontrastu.....65

Obr. 26: Funkce pro snížení kontrastu v obraze 65

Obr. 27: Originální obraz, bez filtrací. 67