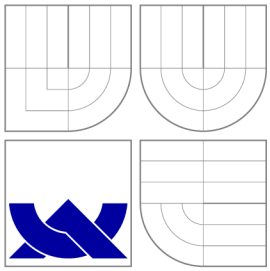
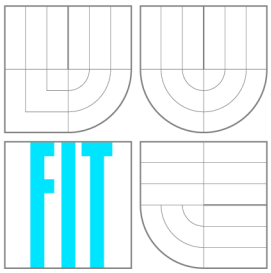


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

VÝUKOVÝ PROGRAM PRO DEMONSTRACI PRINCIPU BAREV A BAREVNÝCH MODELŮ

EDUCATION COMPUTER PROGRAM FOR DEMONSTRATION OF COLORS AND COLORS MODELS PRINCIPLES

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PETR ŠTĚPÁN
AUTHOR

VEDOUCÍ PRÁCE — doc. Ing. PŘEMYSL KRŠEK, Ph.D.
SUPERVISOR

BRNO 2009

Abstrakt

Tato práce se zabývá principy využívanými v souvislosti s barevnými paletami a redukcí barevného prostoru obrazových dat v počítačové grafice. Zvláštní důraz byl kladen na jejich vysvětlení pro výukové účely. Má za cíl pomoci i člověku neznalému těchto algoritmů osvětlit jejich principy a použití. Program za tímto účelem napsaný v jazyce Java slouží jako demonstrační prostředek ke splnění těchto cílů.

Abstract

This thesis describes the principles used when working with color palettes and with the reduction of color space of images in computer graphics. Special attention has been paid to their explanation for educational purposes. It should help even a user not familiar with these algorithms better understand their principles and application. A computer program written in the Java language has been designed to demonstrate and fulfill these goals.

Klíčová slova

barva, barevná paleta, redukce barevného prostoru, dithering, výuka, Java

Keywords

color, color palette, reduction of color space, dithering, education, Java

Citace

Petr Štěpán: Výukový program pro demonstraci principu barev a barevných modelů, bakalářská práce, Brno, FIT VUT v Brně, 2009

Výukový program pro demonstraci principu barev a barevných modelů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Přemysla Krška Ph.D.

.....
Petr Štěpán
11. května 2009

Poděkování

Chtěl bych tímto poděkovat vedoucímu své bakalářské práce, panu doc. Ing. Přemyslu Krškovi, Ph.D. za odborné vedení a cenné rady při tvorbě této práce.

Obsah

Kapitola 1

Úvod

Cílem této bakalářské práce je vytvořit program, který by mohl sloužit k výuce principů redukce barev v počítačové grafice a algoritmů pro práci s nimi. Kladu si za zásadní prioritu celého problému didaktickou stránku věci. Uživatelé tohoto programu by se měli při jeho používání naučit a porozumět způsobům, jakých se používá v počítačové grafice pro práci s barvami a jejich paletami. Problém většiny současných programů, kde se nějakým způsobem pracuje s barvami, je ten, že metody, které používá, nejsou pro běžného uživatele zcela transparentní a tudíž k nim ve valné většině lidé přistupují jako k jakýmsi černým skříňkám a nedokáží si udělat představu o tom, jak pracují algoritmy v nich skryté. Touto prací bych chtěl přispět k vyšší míře schopnosti porozumět těmto nejelementárnějším principům i běžným uživatelům informační techniky. Toho bych chtěl dosáhnout pomocí určité míry interaktivity programu, kdy má uživatel možnost měnit parametry barevného převodu i v jeho průběhu, například tím, že si uprostřed animace může vypnout, či zapnout distribuci chyby nebo změnit práh u prahování a tak porovnat dopad těchto jeho změn v rámci jednoho obrázku.

Tato práce je rozdělena na několik částí. První kapitola je věnovaná teoretickému rozboru dané problematiky, včetně možností jejího uplatnění v praxi. V další kapitole se věnuji návrhu programu, zejména jeho uživatelského rozhraní, včetně nástinu implementačního řešení některých problémů. Následují kapitoly se zhodnocením výsledků a celkovém přínosu této práce.

Kapitola 2

Teoretický rozbor

2.1 Redukce barevného prostoru

V současné době se setkáváme s obrovským množstvím obrazových dat, nejen v počítačové oblasti, ale i v běžném životě. Tato obrazová data nabývají stále více na objemu a jejich používání se stává stále větší samozřejmostí. Proti sobě zde stojí dvě protikladné tendence. Na jedné straně je to snaha o barevně co nejvěrnější zachycení skutečnosti - například u digitální fotografie. Na straně druhé je požadavek na co nejmenší objem obrazových dat. Pro hledání kompromisu mezi těmito dvěma tendencemi se můžeme vydat několika různými cestami, například hledáním co nejlepšího kompresního algoritmu pro uložení těchto dat.

Cílem této bakalářské práce je ovšem popsat takové principy, při nichž dochází k úspoře ukládaných dat pomocí zredukování barevného prostoru obrázku. Například v počítačové grafice běžně používaná paleta Truecolor [11] (16 777 216 barevných odstínů) používá pro uložení informace o barvě jednoho bodu 24 bitů (8 bitů pro červenou složku, 8 pro zelenou a 8 pro modrou). Pokud použijeme vhodné postupy a zredukujeme daný obrázek s distribucí chyby na 8 bitů (paleta 3 - 3 - 2, 256 barevných odstínů), získáme relativně kvalitní obrazový výstup s třetinovým datovým objemem. V praxi se hojně setkáváme s požadavky na používání těchto technik, ať už z optimalizačních důvodů (úspora místa při zachování celkového dojmu z obrazu), nebo je jejich nasazení přímo nezbytné (například v tiskárnách). Dvěma hlavními postupy v této souvislosti jsou převod obrázku ve stupních šedi na černobílý obraz a redukce palety u barevných obrázků. V následujících kapitolách se budu podrobněji zabývat těmito oblastmi, včetně vysvětlení používaných algoritmů. Ještě před tím bude řečeno něco o převodu na stupně šedi.

2.2 Převod na stupně šedi

Nejelementárnějším způsobem převodu barevné škály obrazových dat je převod z barevné palety na paletu odstínů šedi. Tento triviální algoritmus se provádí buďto sám o sobě na místech, kde je to třeba, ale hlavně jako mezikrok ve všech případech, kdy požadujeme znát světelnou intenzitu bodu. Nachází tedy uplatnění vždy, když převádíme barevný obrázek na jednobitovou černobílou paletu. Tomuto problému se budeme věnovat dále.

Každý pixel barevného obrázku ve stupnici RGB má tři základní složky, červenou, zelenou a modrou (dále v textu budu používat označení R pro červenou složku, G pro zelenou a B pro modrou). Tyto hodnoty jsou na sobě nezávislé a jejich kombinací dostáváme odstín dané barvy. Mějme například obrázek s barevnou hloubkou 24 bitů, tudíž 8 bitů (256

Obrázek 2.1: Vlevo je původní obrázek, vpravo převedený na stupně šedi

odstínů) pro každou barevnou složku. Z tohoto obrázku budeme chtít vytvořit bitmapu o 256 stupních šedi. Ze všeho nejdříve potřebujeme získat světelnou intenzitu daného pixelu, tu budeme značit I. Známe všechny tři barevné složky bodu, tudíž je můžeme aplikovat do jednoduchého empirického vzorce:

$$I = 0.299 * R + 0.587 * G + 0.114 * B$$

Tento vzorec vychází z empirického studia citlivosti vnímání lidského oka na jednotlivé barevné složky a výsledný obraz nejvěrněji kopíruje vstupní obrázek.

Nyní máme hodnotu světelné intenzity daného bodu, a tuto hodnotu uložíme jako červenou, modrou i zelenou složku výsledného pixelu. Pro odstíny šedi totiž platí, že $I = R = G = B$. Výsledek tohoto převodu je vidět na obrázku 2.1.

2.3 Převod na černobílý obraz

S tímto přístupem se setkáváme v praxi velice často. Jedná se o situaci, kdy máme obrázek ve stupních šedi (případně obrázek barevný, který si triviálně na stupně šedi převedeme) a potřebujeme tuto obrazovou informaci zredukovat pouze na dvoubarevnou paletu, resp. černou a bílou barvu. Toto je velice praktické, jednak kvůli datové nenáročnosti (na uložení jednoho pixelu nám stačí jeden bit), a také díky maximální možné rychlosti následného zpracovávání obrazu. Proto je také této redukce využíváno při druhotném zpracování informací (například při rozpoznávání objektů). Existují také případy, kdy kvůli technickým omezením média nemáme ani jinou možnost reprezentace obrazových dat, například černobílé tiskárny či grafické monochromatické LCD displeje.

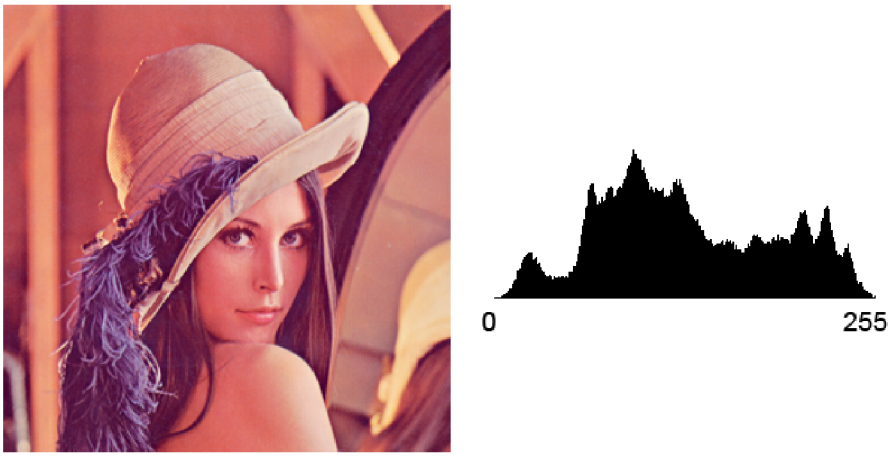
Tak masivní zredukování barevné palety ale v sobě nese také obrovské množství ztracené barevné informace. Výše uvedená využití této techniky se podstatnou měrou liší v tom, do jaké míry u nich ztrátovost informací vadí: například u počítačového zpracování obrazu a rozpoznávání určitých objektů (potažmo jejich hran [3]). V něm nám jde o kontrasty, tudíž nám tato ztráta tolik nevadí. Proto se pro tento přístup hodí použití prahování. Otázkou u konkrétního využití zůstává, jak zvolit co nejlepší práh.

Naopak, pokud nám jde o kvalitu výsledného obrazu a snažíme se nějakým způsobem kompenzovat onu zmiňovanou ztrátu informace, je pro nás lepší použít jinou metodu redukce. Například v tiskárnách se velice často využívá metoda maticového rozptýlení (v podobě halftoningu), která při vhodně zvolené rozptylové matici dává velice kvalitní tiskový výstup. Je možné také použít distribuci chyby u některé z variant (nejčastěji u prahování) a tak významnou měrou využít i část ztrátových informací při průběhu redukce. Celkový přehled všech používaných metod je možné shlédnout zde [2].

Nyní se podíváme podrobněji na několik hlavních způsobů převodu obrazu na jednobitovou paletu.

2.3.1 Prahování

Prahování (anglicky thresholding [10]) je v podstatě nejelementárnějším přístupem k problému redukce obrazu do dvou barev. Jeho princip je takový, že nejprve u každého vstupního pixelu spočítáme jeho intenzitu. V případě obrázku ve stupních šedi odpovídá intenzitě hodnota kterékoliv barevné složky (červené, zelené, či modré). U barevného obrázku je situace o něco složitější (podrobněji je tato problematika popsána v sekci 2.2, kde se věnuji redukci z barevného obrázku na paletu stupňů šedi). V rámci určitého zjednodušení a snažší demonstrace budu v této práci i v programu pracovat s rozpětím 256 barev. V případě stupňů šedi je to tedy rozpětí intenzit od hodnoty 0 (černá barva) až po hodnotu 255 (bílá barva). V konečném důsledku je tedy každý pixel reprezentován jednou hodnotou intenzity v intervalu $< 0, 255 >$.



Obrázek 2.2: Obrázek s jeho histogramem rozložení intenzit

Nyní si zvolíme prahovou hodnotu této intenzity, podle které budeme rozlišovat, jaká bude hodnota výsledného pixelu. Nevhodné zvolení této hodnoty má za následek až úplnou nečitelnost obrázku (viz obrázek 2.3). Ideální způsob zvolení prahové hodnoty představuje vyčtení střední hodnoty z histogramu obrázku. Na obrázku 2.2 je příklad jednoho obrázku a jeho histogramu intenzit. Čím více bychom šli doleva, tím více tmavých odstínů (nižších světelných intenzit) obrázek obsahuje, naopak vpravo jsou tóny světlejších barev.

Na tomto ukázkovém případě vidíme, že ideální hodnoty pro výběr prahu budou ležet zhruba někde uprostřed, což demonstruje levá část obrázku 2.3. Pokud bychom zvolili práh blíže k extrémům (například hodnotu 200 - viz pravá část obrázku 2.3), výsledky jsou již značně nedostatečné.

Obrázek 2.3: Prahování s vhodně (128) a nevhodně (200) zvoleným prahem

V praxi existují i další metody, jak získat hodnotu prahu, například adaptivním prahováním, které bere v potaz okolní body při dynamickém výpočtu prahu atd.

Jakmile máme vybranou hodnotu prahu, se kterou chceme upravit vstupní obrázek, tak můžeme začít samotný proces redukce. Ten spočívá v tom, že na každý pixel obrázku aplikujeme následující rovnici:

$$f(x,y) = \begin{cases} 0, & l(x,y) < T \\ 1, & l(x,y) \geq T \end{cases}$$

Funkční hodnota funkce $f(x,y)$ v bodě o souřadnicích x a y značí výslednou hodnotu pixelu po redukci, což může být buď 0 (černá barva), nebo 1 (bílá barva). Funkce $l(x,y)$ značí světelnou intenzitu daného bodu a T je hodnota prahu, se kterou pracujeme. V podstatě nám tento funkční předpis říká, že pokud je intenzita aktuálního pixelu větší nebo rovna prahové hodnotě, tak je jeho výsledná barva bílá, pokud je menší, než práh, tak výsledná barva je černá.

Metoda prahování má v různých obměnách široké využití v počítačové grafice. I všechny následující postupy redukce na jednobitovou monochromatickou paletu, které budu popisovat, jsou principielně velmi podobné prahování, liší se jen způsobem výběru prahové hodnoty.

2.3.2 Náhodné rozptýlení

V předchozím případě jsme měli jasně danou a předem vybranou prahovou hodnotu, vůči které jsme porovnávali hodnoty intenzit všech bodů v obrázku. U náhodného rozptýlení zaneseme do tohoto algoritmu trochu nedeterminismu, přičemž dostaneme celkem zajímavé výsledky. Samotný postup spočívá v tom, že použijeme generátor náhodných čísel (například lineární kongruentní generátor - viz. [7]), pomocí něhož získáme pro každý pixel náhodné číslo z intervalu $< 0, 255 >$. To se následně použije jako prahová hodnota pro daný pixel, jak je to popsáno v části o prahování 2.3.1.

Tato metoda je velice jednoduchá a rychlá. V případě, že pracujeme s obrazovými daty o větším rozlišení a s většími kontrastními plochami, je tato metoda až překvapivě účinná a produkuje relativně velmi dobré výsledky. Na základě pravděpodobnostních modelů pak

Obrázek 2.4: Obrázek zredukovaný pomocí náhodného rozptýlení

vyplývá důležitá vlastnost tohoto způsobu barevné redukce, kterou je zachování jasových poměrů jednotlivých částí obrazu. Pokud máme obrázek s dostatečně velkým rozlišením, jeví se tato možnost jako velmi dobré řešení.

2.3.3 Maticové rozptýlení

Ještě kvalitnější výstup nám při vhodně zvolených parametrech dává metoda maticového rozptýlení. Klíčovou roli v tomto případě hraje distribuční (rozptylová) matice. Samotný princip této metody je popsaný v knize Moderní počítačová grafika [12]:

> Maticová rozptylovací metoda používá pro generování různých odstínů šedi pravidelných, předem daných vzorků složených z bílých a černých bodů. Pokud jsou tyto vzorky dobře navrženy, působí výsledný obraz jako kresba, ve které malíř tužkou vystínoval tmavší, či světlejší místa pravidelnými jemnými šrafami.

Samotná distribuční matice je libovolně velká matice různě rozmístěných hodnot z řady 0 až n-1, kde n je celkový počet prvků dané matice. Žádný z jejich prvků se neopakuje. Nejčastěji používaná je matice o velikosti 4 x 4, tzn. celkem 16 prvků. Stejnou matici používám i v této bakalářské práci. Právě rozmístění jejích prvků je klíčové pro celkový výsledný dojem zredukovaného obrazu. Je mnoho možností, jak rozmístit oněch 16 čísel v rámci matice, ne všechny kombinace však vykazují dobré výsledky. Jde o to, že maticový rozptyl je založený na určité pravidelnosti rozmístěných bodů ve výsledném obrazu. Tuto

pravidelnost, potažmo i celkový vjem, ovlivníme různým rozmístěním prvků této matice. Více o této problematice je možno najít na [9].

Jako příklad takovéto matice uvedu matici M_d [13], která se používá nejčastěji pro účely zobrazování výsledného obrazu:

$$M_d = \begin{vmatrix} 0 & 12 & 3 & 15 \\ 8 & 4 & 11 & 7 \\ 2 & 14 & 1 & 13 \\ 10 & 6 & 9 & 5 \end{vmatrix}$$

Tato matice se vyznačuje tím, že vytváří pravidelné obrazce bodů specificky se soustředící kolem jejího středu, což má za následek iluzi více či méně šedých ploch, stejně jako ve vstupním obrázku. Dalším příkladem může být distribuční matice s označením M_p [13]:

$$M_p = \begin{vmatrix} 1 & 5 & 9 & 2 \\ 8 & 12 & 13 & 6 \\ 4 & 15 & 14 & 10 \\ 0 & 11 & 7 & 3 \end{vmatrix}$$

Tato nachází uplatnění při tiskovém výstupu obrazu, především proto, zejména z důvodu pro ni charakteristického vytváření shluků černých bodů. Tyto shluky vyvolávají při dostatečném rozlišení dojem celistvého obrazu. V praxi jsou využívány právě při tisku, kdy se například u inkoustových tiskáren mnohem snáze tisknou větší shluky, než samostatné body. Také je využíváno mírné rozpití kapiček inkoustu do papíru kolem shluku, což přispívá k větší výsledné jemnosti a kvalitnějšímu dojmu vytištěného obrazu.

Obrázek 2.5: Obrázky zredukované pomocí matic M_d a M_p

Distribuce pomocí rozptylové matice se dá rozdělit na dva základní způsoby používání: halftoning (polotónování) a dithering (rozptýlení). Nyní si vysvětlíme jejich princip.

Halftoning

Tato technika je dříve navržená a používaná už v 19. století, kdy byla takto vytisknuta v roce 1872 první fotografie v novinách [5]. V oblasti počítačové grafiky se polotónováním myslí

specifické použití distribuční matice (nejčastěji právě výše uvedená matice M_p). Každý pixel vstupního obrazu se nahradí plochou 16 pixelů (při použití matice 4 x 4), kde bude vyplněna jejich určitá část černou barvou a zbylé barvou bílou. O tom, jaký obrazec bude nahrazen namísto vstupního pixelu, rozhoduje jeho světelná intenzita. Tato hodnota se porovnává s hladinami určenými danou maticí. Pokud je přesáhnuta hladina pro daný prvek matice, bude onen subpixel bílý, v opačném případě černý.

Obrázek 2.6: Obrázek získaný za pomocí metody halftoningu, upravený na původní velikost

Lépe toto bude ilustrovat následující příklad. Vezměme si první řádek matice M_p, resp. jeho hodnoty 1, 5, 9, 2. Pracujeme s 256 stupni intenzity a matice má 16 prvků, tudíž prahové hodnoty pro tyto prvky vypočítáme jako $(256/16) * i$, kde i je hodnota uvedená v matici. Když si tyto hodnoty dosadíme do výše uvedeného vzorce, tak pro první řádek matice M_p nám vycházejí prahové hladiny 16, 80, 144, 32. Předpokládejme, že vstupní pixel měl světelnou intenzitu 50. Tato hodnota je vyšší, než první a poslední hladiny v našem příkladě. Proto budou tyto prvky reprezentovány bílými subpixely. Naopak prostřední dvě hladiny jsou vyšší, než intenzita vstupního pixelu, tudíž budou tyto dva subpixely černé. Analogicky postupujeme pro celou matici a všechny pixely vstupního obrázku nahradíme skupinou subpixelů s obrazcem odpovídajícímu dané matici. Je logické, že tímto přístupem se rozlišení obrázku čtyřikrát zvětší, avšak jeho kvalita a zejména tiskový výstup bude velmi dobrý.

Dithering

Druhou technikou využívající distribuční matici je barevné rozptýlení. Touto technikou se zabývá i mnou implementovaný program k tomuto tématu. Výše popsaný systém hladin jednotlivých prvků matice a jejich porovnávání se světelnou intenzitou se aplikuje i v tomto případě, avšak s jedním rozdílem. Matice se neaplikuje místo jednoho pixelu, ale jakoby dlaždicově se rozloží po celém obrázku a každá hladina se porovnává s intenzitou odpovídajícího pixelu vstupního obrazu. V praxi to poté vypadá tak, že se provede operace modulo nad x-ovou souřadnicí obrázku s číslem 4 (v případě matice 4 x 4) a výsledné číslo z intervalu $< 0, 3 >$ nám udává x-ový index prvku v matici. Totéž uděláme pro y-ovou souřadnici a získáme tak přesný index prvku v matici, ze kterého vytvoříme hladinu, kterou porovnáme s intenzitou vstupního pixelu. Pokud je tato prahová hodnota nižší než intenzita vstupního pixelu, pak bude odpovídající výstupní pixel bílý a naopak.

V tomto případě zůstává rozlišení výchozího obrazu stejné, jako u obrazu vstupního, proto se tato metoda používá v počítačové grafice více než předešlá. Při použití matice M_d vykazuje tato metoda výsledný obrazový vjem nejvíce věrný předloze ze všech tří dosud uvedených metod redukce barevného prostoru na jednobitovou monochromatickou paletu. Zdaleka nejvěrnější výsledný obraz dostaneme až při použití distribuce chyby s některou z těchto metod, čemuž se budeme věnovat podrobněji v následující fázi.

2.3.4 Distribuce chyby

Obrázek 2.7: Metoda prahování bez a s použitím distribuce chyby (se stejnou hodnotou prahu)

Distribuce chyby je nejpokročilejší technika při převodu obrázku na monochromatickou paletu. Umožňuje při redukci brát v potaz i okolní body pixelu a část informace, která se ztratila při navzorkování předchozích bodů na černou nebo bílou barvu. Pro tuto metodu není podstatné, jakým algoritmem jsme se dobrali k tomu, že daný bod bude bílý, nebo černý. Právě proto se výborně hodí jako doplněk ke všem předchozím (i následujícím) metodám. Princip je takový, že pokud máme vstupní pixel o intenzitě 128 a jakoukoliv dříve předvedenou metodou rozhodneme, že výsledná barva bude černá (intenzita 0), dopustili jsme se chyby o hodnotě 128, o kterou jsme zmenšili intenzitu výstupního pixelu. Informaci o této chybě musíme roznést mezi okolní pixely.

K tomuto účelu se používá několik způsobů, z nichž všechny jsou založeny na tzv. chybové distribuční matici. Každý přístup se liší právě použitou maticí, my si zde ukážeme Floyd-Steinbergovu distribuční matici (celkový přehled se dá najít zde [2]).

$$\begin{vmatrix} 0 & 0 & 0 \\ 0 & X & 7 \\ 3 & 5 & 1 \end{vmatrix}$$

Z aktuálního pixelu, který je reprezentován prostředním prvkem v matici (označen X), vypočteme chybu vzniklou při redukci (pokud je výsledný pixel bílý, použijeme vzorec $E = I - 255$, pokud černý $E = I$, kde E je výsledná chyba a I je intenzita aktuálního pixelu). Tuto chybu poté rozdělíme mezi sousední pixely podle daného koeficientu v matici. K intenzitě bodu sousedícímu zprava přičteme $\frac{7}{16}$ hodnoty chyby, k intenzitě pixelu pod ním $\frac{1}{16}$, k bodu pod aktuálním pixelem $\frac{5}{16}$ a k pixelu s ním vlevo sousedícím $\frac{3}{16}$ chyby. Chyba se rozdistribuovává přidáním či odečtením dané hodnoty od příslušného pixelu, při zjišťování jeho intenzity. Prakticky se toto dá implementovat přidáním chybové matice k obrázku, která bude mít stejný počet prvků, jako má vstupní obraz rozlišení.

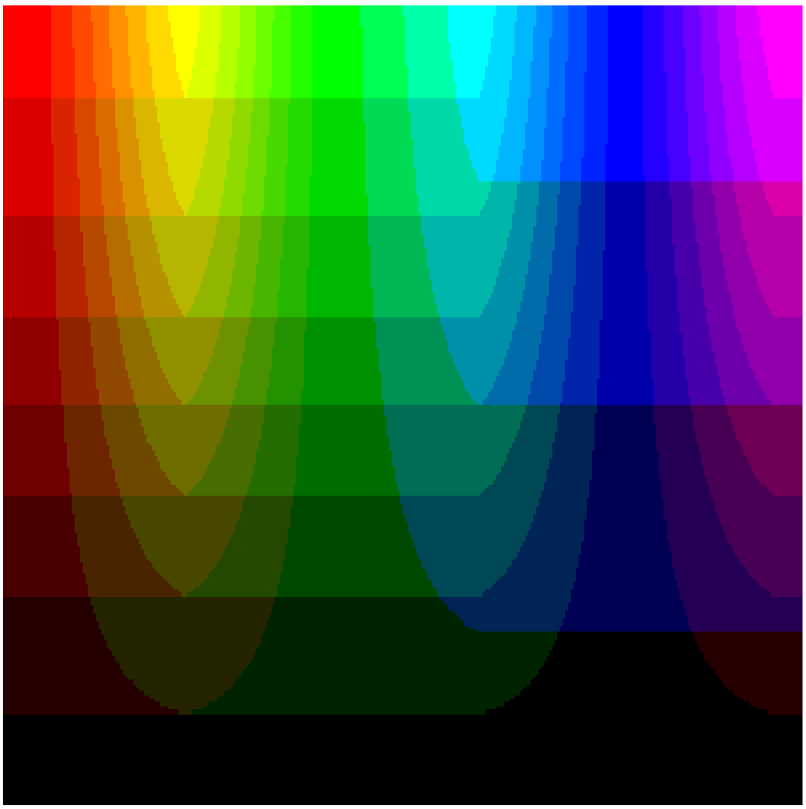
Zvláštním případem přenášení chyby je barevná distribuce. Ta se používá v případě, že chceme distribuovat nejen chybu intenzity, ale i chyby jednotlivých barevných složek. Proto se používá tehdy, když výstupem je barevný obrázek (převod do palety 3 - 3 - 2, převod do palety false colors, atd.). Chyba každé barevné složky se vypočítá jako $E_R = R_{in} - R_{out}$, kde R_{in} je hodnota vstupní barevné složky (v tomto případě červené) a R_{out} složka výstupní (vypočítaná podle zvolené metody). Tímto způsobem dostaneme 3 hodnoty v teoretickém rozmezí -255 až +255, které podobným způsobem jako v předchozím případě zohledňujeme při zpracovávání okolních bodů.

2.4 Redukce barevné palety

Další sekcí týkající se redukce barevného prostoru je redukce barevné palety obrázku. Tímto pojmem se rozumí situace, kdy máme obrázek o určité barevné hloubce a potřebujeme omezit počet jeho barev. K tomu nás může vést buď potřeba úspory datového objemu v případech, kdy není potřeba vysoká kvalita obrázku (typicky například obrázky formátu gif či png publikované na webu, které používají adaptivní palety obsahující jen odstíny, které jsou v obrázku obsaženy), nebo je nutnost barevné redukce zapříčiněna zobrazovacím zařízením, na kterém bude grafika prezentována. Příkladem jsou některé starší monitory nebo zařízení schopná zobrazit jen několik barev (starší mobilní telefony s barevným displejem dokázaly zobrazit mnohdy jen třeba 256 barev). Dále je možné rozdělit tyto metody na případy, kdy pouze snižujeme barevnou hloubku obrazu (převod na paletu 3 - 3 - 2, převod na osmibarevnou paletu), nebo kdy měníme i celkové barevné vzezření obrázku (paleta false colors, gradient).

2.4.1 Paleta 3 - 3 - 2

Běžná barevná paleta truecolors má barevnou hloubku 24 bitů (8 bitů pro každou složku). Velmi často se stává, že potřebujeme reprezentovat daný obrázek pouze v 8 bitech na každý pixel. Za tímto účelem můžeme použít právě paletu s označením 3 - 3 - 2. Tato paleta je charakteristická tím, že nepokrývá výtažek ze všech barevných odstínů, ale pouze z těch, na které je lidské oko nejcitlivější. Proto je místo určené k uložení jednoho pixelu rozděleno na



Obrázek 2.8: Obrázek převedený do palety 3 - 3 - 2, můžeme si všimnout menšího množství modrých odstínů

3 bity pro červenou barevnou složku, 3 bity pro zelenou a pouze 2 bity pro složku modrou. Důsledek tohoto je, že paleta umí zobrazit menší množství modrých odstínů, což je patrné i na obrázku 2.8.

Z toho nám vychází, že tato paleta umí zobrazit celkem 256 barev. Ona osmibitová hodnota, kterou je daný pixel charakterizován, je vlastně indexem do této palety, pod kterým se v ní nachází příslušný barevný odstín. Podrobnější způsob vytvoření palety a hledání indexu pro konkrétní pixel je blíže popsán v kapitole 3.7.

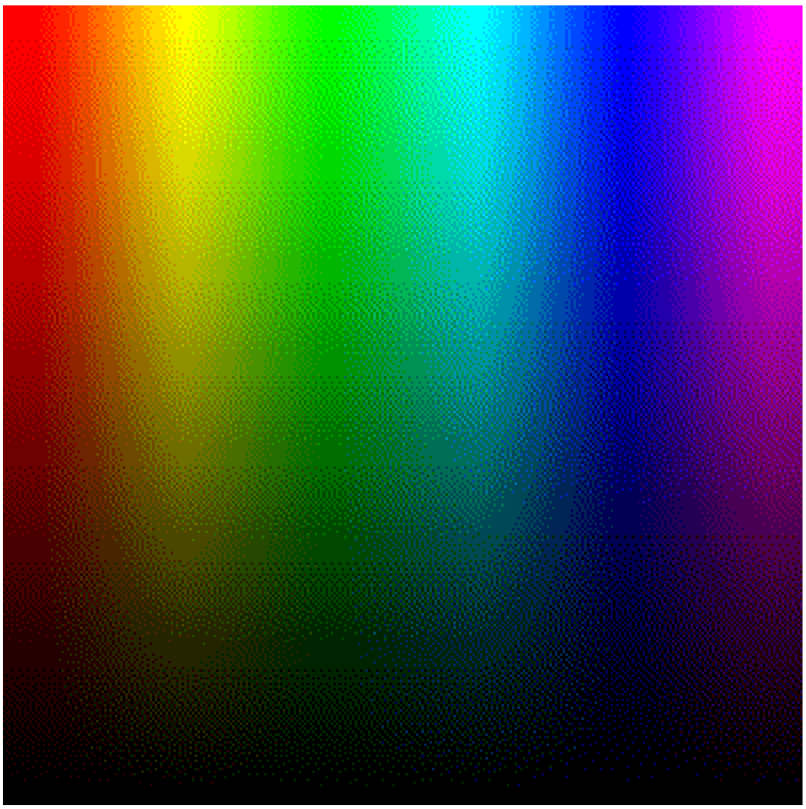
I zde můžeme využít distribuci chyby, která zajistí mnohem jemnější přechody mezi barvami.

2.4.2 Speciální barevné palety

V této sekci se pouze okrajově zmíním o dalších zajímavých barevných paletách, které se používají v počítačové grafice.

Paleta false colors

Jedná se o zvláštní druh palety, používaný například v astronomii, termografii, či v lékařském zobrazování. Obecný princip je takový, že namapujeme určitou barevnou škálu na veličinu, kterou chceme pozorovat. V mnou implementovaném případě se jedná o barevnou škálu světelného spektra aplikovanou podle intenzity vstupního pixelu. Názornější příklad



Obrázek 2.9: Stejný obrázek jako 2.8 v paletě 3 - 3 - 2, tentokrát však s distribucí chyby, jsou na něm patrné mnohem jemnější barevné přechody

je na obrázku 3.8.

Mezi další praktické případy využití principu této palety „falešných“ barev patří termovize, kde je barevná škála využita ke grafickému znázornění množství infračerveného záření vydávaného okolním prostředím. Ještě běžnější je její použití v meteorologii, konkrétně u radarových snímků, kde její odstíny reprezentují množství oblačnosti nad určitým územím, viz. obrázek 2.11.

Jak je vidět, použité barevné škály se od sebe mohou lišit, proto neexistuje jednotná paleta false colors (jako například paleta RGB). Jedná se o obecný popis principu, který má svoje specifické použití a využití.

Gradient

Toto není paleta v pravém smyslu slova, jedná se o přechod mezi dvěma barvami. Její princip je opět založený na rozhodování podle světelné intenzity vstupního bodu, stejně jako u palety false colors. Vybereme si první barvu, která bude reprezentovat body s nejmenší intenzitou, druhá naopak s intenzitou nejvyšší a všechny pixely s hodnotami intenzity někde mezi nimi budou ve výsledku odpovídat příslušnému odstínu vzniklého prolnutím obou původních barev.

Výsledky tohoto přístupu můžete porovnat na obrázku 2.12.

Speciálním případem gradientu je obrázek ve stupních šedi, kde prolínanými barvami jsou černá a bílá.

Obrázek 2.10: Vlevo původní obrázek, vpravo po převodu na paletu false colors

RGB palety s omezenou barevnou hloubkou

Další paletou, z uvedených asi nejčastěji používanou, je prosté omezení barevné hloubky, kdy se každá z barevných složek rovnoměrně zredukuje na menší prostor. Například 3 bitová paleta RGB je schopna zobrazit 2^3 čili 8 barev. Tato technika je standardizována u televizního teletextu.

Stejně tak je možné vytvořit 6 bitovou paletu, kdy na každou barevnou složku připadají 2 bity, tudíž je schopna zobrazit 64 barevných odstínů. Tato paleta byla použita u standardu pro zobrazování EGA (Enhanced Graphics Adapter), který měl své využití u počítačů IBM PC-AT [4].

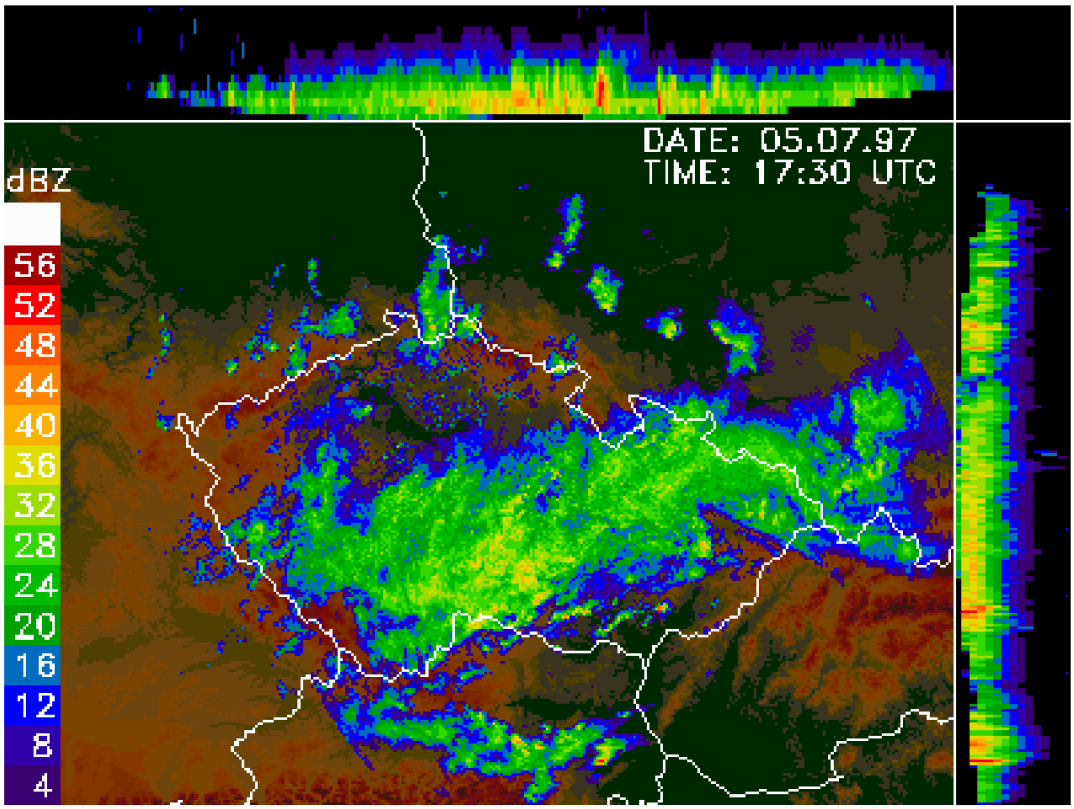
Takovýchto palet existuje celé množství, jejich výčet je uveden [8].

2.5 Problematika výuky algoritmů pro práci s barvami

Všechny algoritmy pro redukci barevného prostoru jsou jednoduché a i jejich principy jsou snadno pochopitelné. Problém tkví v tom, že si mnohdy nedokážeme představit, jak vlastně bude vypadat výsledný obrázek. Použijme například algoritmus náhodného rozptýlení (3.3). Tento velice jednoduchý princip je založený na kombinaci prahování a náhodného výběru prahu. Je až překvapivé, jakých tento postup dosahuje výsledků, navzdory tomu, že se nám může zdát ona náhodnost jako zdroj chaosu a že výsledný obrázek nemůže být čitelný.

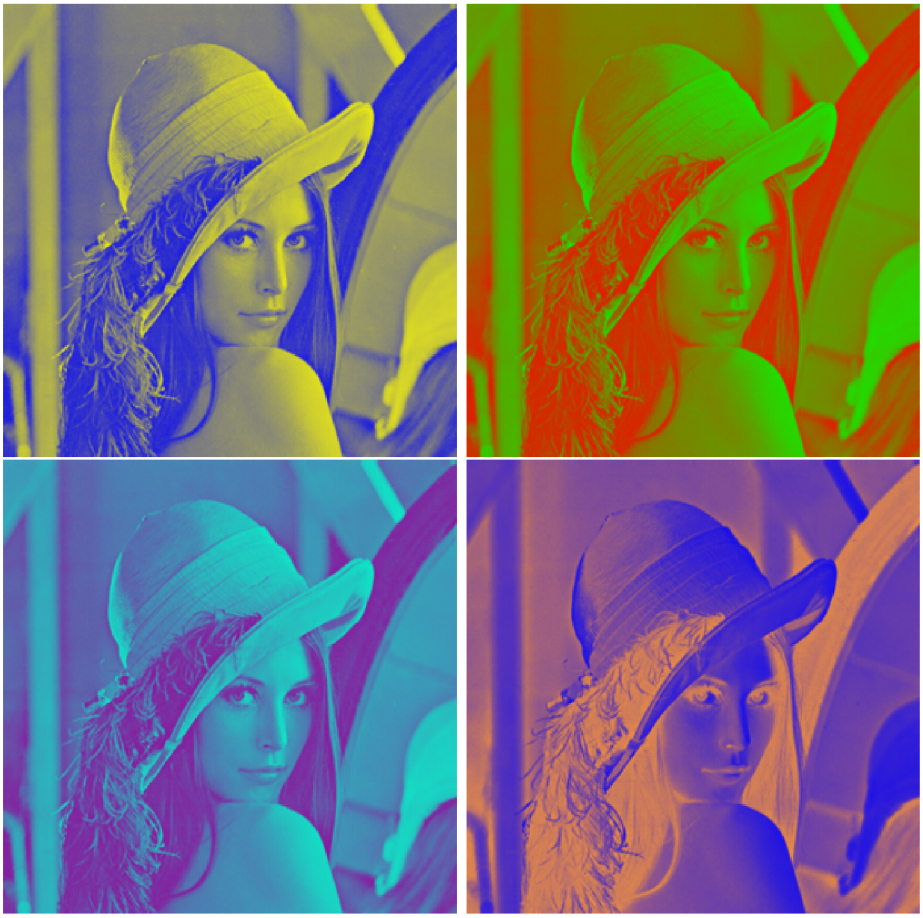
Z osobní zkušenosti vím, že uživatel zajímající se o danou problematiku a nemající ještě mnoho zkušeností, by uvítal právě takový druh programu, kde by mohl nejen hned prezentovat výsledek daného principu, ale kde by si mohl i spustit animaci s podrobnějšími informacemi o jednotlivých krocích. Zároveň by si v jejím průběhu mohl měnit vstupní parametry a mohl by tak porovnávat vliv jejich změny na kvalitu výsledného obrázku.

Tento styl výuky je díky své interaktivitě mnohem přínosnější pro rychlejší a lepší pochopení daných postupů, než pokud člověk vidí jen jejich popis v učebnici, či jen výsledek na obrázku nebo v grafickém editoru. Velmi důležitou součástí je také možnost nahrávat si libovolné obrázky a provádět s nimi tyto operace. Takto můžeme zjistit spoustu zajímavých informací o jednotlivých postupech, které by nám mohly zůstat skryty, pokud uvidíme jen obrázek v učebnici, ze kterého není konkrétní detail dostatečně patrný.



Obrázek 2.11: Použití modifikované palety false colors pro reprezentaci dat z meteorologického radaru z Českého hydrometeorologického ústavu [1]

Díky všem těmto faktům si myslím, že je nesmírně důležitá nejen existence programů, které umí daný algoritmický postup zpracovat, ale také programů, které jej dokáží řádně popsat a napomůžou tak k jeho lepšímu pochopení. Právě obohacení této skupiny programů jsem si stanovil jako hlavní cíl své bakalářské práce



Obrázek 2.12: Obrázky vzniklé pomocí palet různých barevných přechodů

Obrázek 2.13: Redukce barevného prostoru na 8, resp. 64 barev

Kapitola 3

Návrh a implementace

V této kapitole se podrobněji zaměříme na vznik programové části, od rané fáze návrhu až po detaily implementace. Každá sekce bude věnována jedné části programu, která bude demonstrovat příslušnou techniku redukce barevného prostoru. Většina sekcí obsahuje jednoduchý grafický návrh uživatelského rozhraní pro daný problém a slovní popis, který dále zpřesňuje funkčnost konkrétní části.

V poslední části této kapitoly se zmíním o výsledném programu, popíšu jeho uživatelské rozhraní a sdělím postup na jeho ovládání.

3.1 Návrh aplikace

Začnu nejdříve objektovým návrhem celé aplikace. UML diagram tříd pro snažší pochopení je zde 3.10.

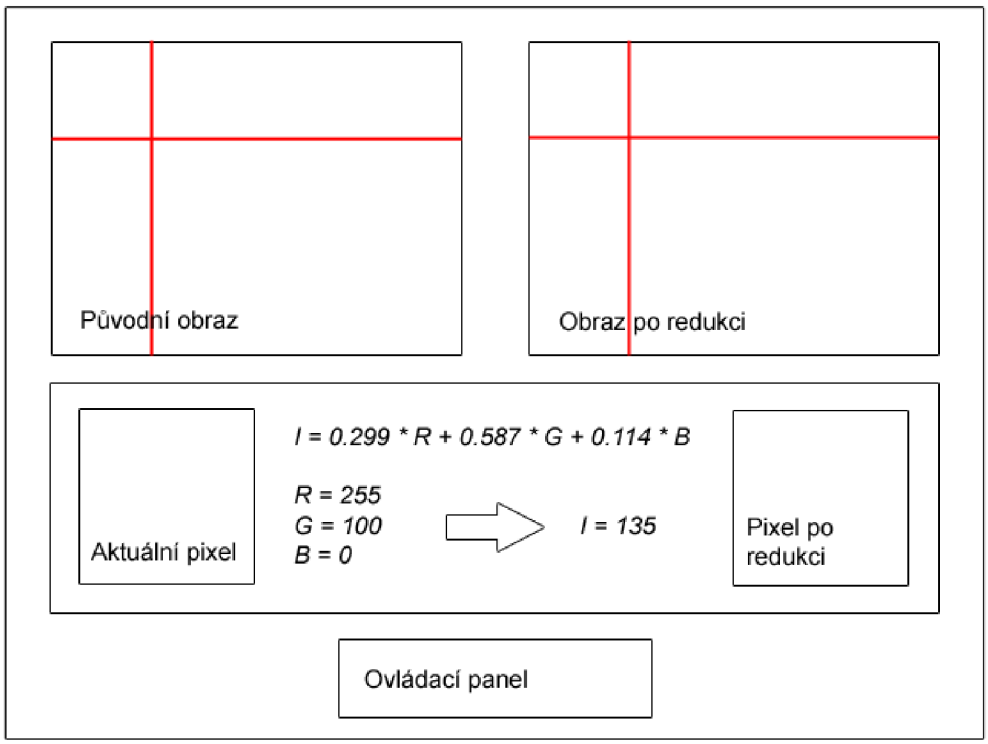
Základem je třída Window. Ta obsahuje základní okno programu s nástrojovými lištami a menu. Rovněž obsahuje metody pro základní ovládání programu. Důležitým atributem této metody je instance třídy Content. Tato třída definuje základní rozložení prvků, které je společné pro všechny obsahy. Z návrhu vyplynulo, že princip všech implementovaných algoritmů z hlediska GUI (grafického uživatelského rozhraní) vykazuje značnou míru podobnosti, co se např. rozložení jednotlivých panelů týče. Proto byl zvolen přístup, v němž máme základní třídu Content, kde jsou již všechny společné prvky nadefinovány, a od které potom dědí všechny třídy s obsahy (Grayscale, RandomDistraction, MatrixDistraction a další). Zároveň jak třída Content, tak všechny třídy od ní dědící, implementují rozhraní Steppable, které obsahuje deklarace metod potřebných ke krokování animace. Tímto je zajištěno, že na úrovni hlavního okna nás nezajímá, jaký obsah je aktuálně zobrazen, ale víme, že díky použitému rozhraní na jeho objekt můžeme volat všechny metody potřebné k běhu animace.

Samotnou animaci má na starosti instance třídy Run, která po jejím spuštění v paralelním vlákně periodicky volá metodu pro provedení kroku daného obsahu. Průběh animace je možno pozastavit, spustit, vrátit do původního stavu, nebo naráz vykonat.

Všechny další potřebné objekty jsou v balíčku tools. Zde jsou třídy definující jednotlivé stavební elementy, jako například třída Image reprezentující panel s obrázkem a obsahující funkce pro práci s barvami, třída Matrix definující rozptylovou matici, apod.

Nyní se zaměříme na funkční a grafický návrh jednotlivých obsahů, včetně implementačních detailů.

3.2 Převod na stupně šedi

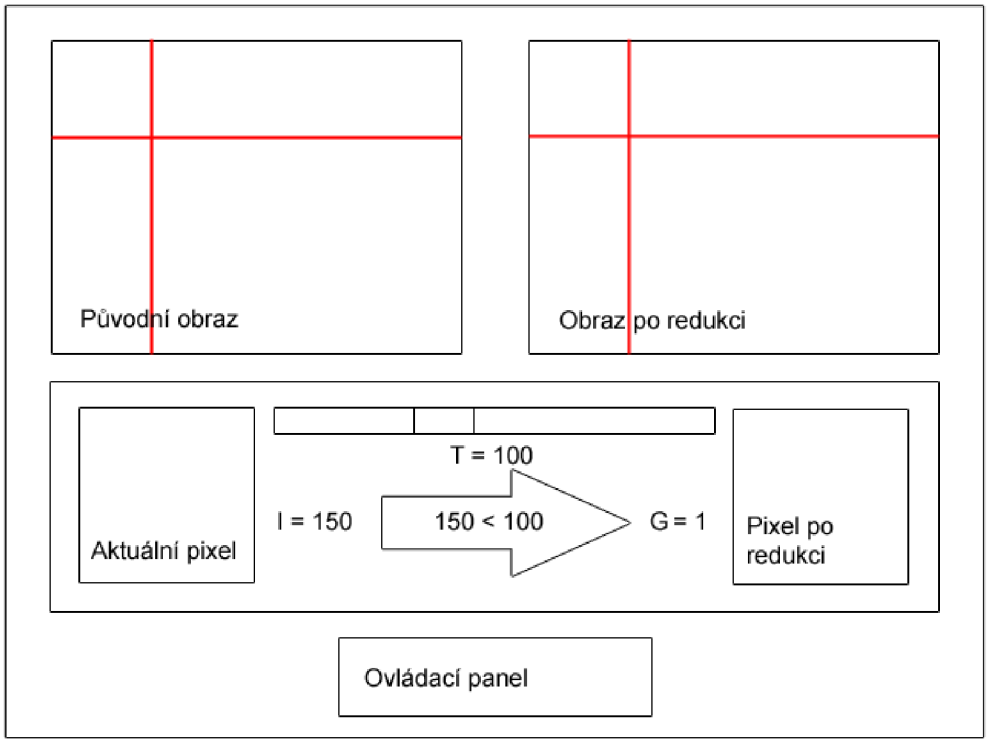


Obrázek 3.1: Převod do stupňů šedi

Základem programu jsou dva obrázky, původní a redukovaný. Původní obrázek je barevný, výsledkem procesu je obrázek převedený do 256 stupňů šedi. Redukovaný obrázek je na počátku pouze prázdná bitmapa, do které se postupně přidávají pixely. Obě bitmapy mají jednoduchý způsob navigace, který pixel v procesu redukce je aktuální (zde červený kříž). Jeden krok probíhá následovně: Navigační kříž se o jeden pixel posune, na místě „Aktuální pixel“ se zobrazí jeho barva v původním obrázku. Vzorec uprostřed je jen informativní text, aby uživatel věděl, podle čeho se počítá. Podle hodnot v aktuálním pixelu se doplní čísla u jednotlivých barevných složek R, G a B. Zároveň se rovnou přepočítá výsledná intenzita I a zobrazí se příslušný odstín šedé barvy do pole „Pixel po redukci“. Tento pixel se také doplní do výsledné bitmapy. V poli „Ovládací panel“ budou prvky pro ovládání procesu barevné redukce. Tento by měl běžet automaticky, ale uživatel by měl mít možnost jej kdykoliv pozastavit a krokovat sám, případně znovu spustit automatický proces. Další možností je zvolit zrychlený průběh bez jakýchkoliv výpisů.

3.3 Prahování

Okno programu je velice podobné, jako v případě prahování, až na to, že vstupní obrázek je ve 256 stupních šedi. Výsledkem procesu bude obrázek převedený do dvou barev - černé a bílé. Hlavní rozdíl je ve středovém panelu. Před samotnou redukcí si uživatel zvolí hodnotu prahu pro převod do černobílého obrazu, se kterou se bude pracovat. Jakmile spustí proces, je stále možné práh za běhu měnit a tak pozorovat rozdíly. Uživatel může celý proces



Obrázek 3.2: Prahování

zastavit a vrátit vše do původního stavu.

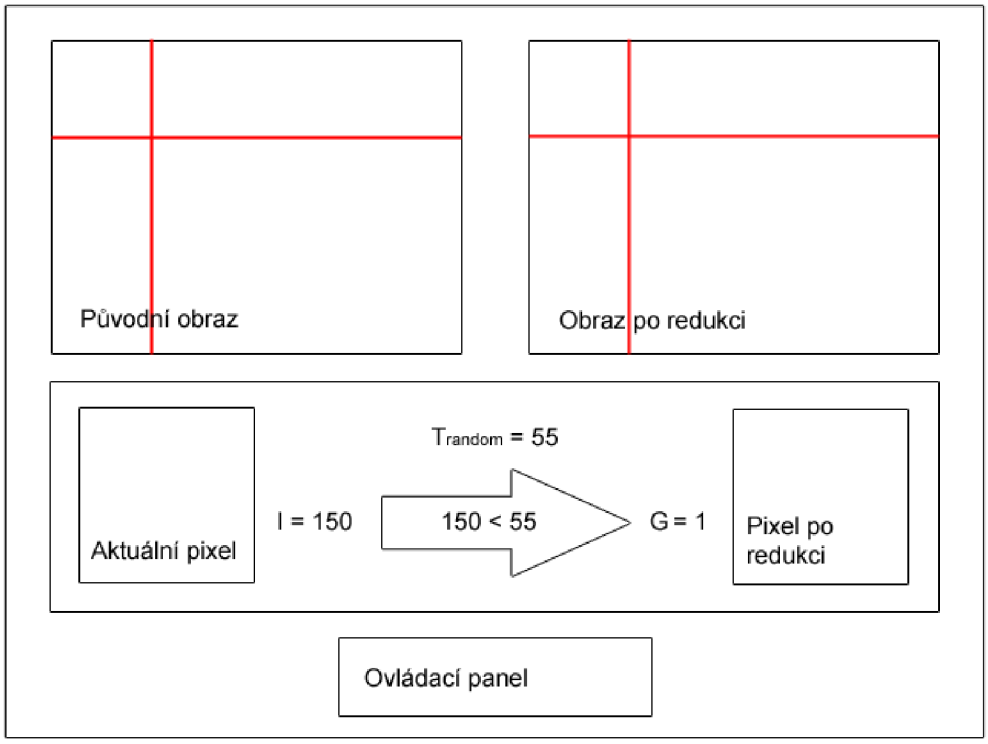
Samotný proces pak bude probíhat tak, že se zobrazí barva aktuálního pixelu, společně s textovou informací o jeho intenzitě. Tato hodnota se porovná se zvoleným prahem a podle výsledku se zvolí hodnota výsledného pixelu G buď 0 nebo 1 (černá, či bílá). Následně se tento pixel uloží na příslušné místo do výsledné bitmapy.

3.4 Náhodné rozptýlení

V případě náhodného rozptýlení vycházíme velkou měrou z výše uvedeného prahování. Algoritmus je ve své podstatě totožný, až na výběr hodnoty prahu pro porovnání s intenzitou aktuálního pixelu. V tomto případě se práh získá náhodnou hodnotou z intervalu 0 až 255 generovanou pro každý pixel zvlášť. Proto se v hlavním panelu okna již nenachází posuvník pro výběr hodnoty prahu, nýbrž se tato hodnota s uniformním rozložením generuje, zobrazí se a s ní je pak porovnávána intenzita a spočítána výsledná hodnota výstupního pixelu.

3.5 Maticové rozptýlení

Zde budeme implementovat redukci na černobílý obraz za pomocí maticového rozptýlení, konkrétně metodou ditheringu. Používat budeme matici 4x4, ve výchozím stavu naplněnou hodnotami odpovídajícími matici M_d.



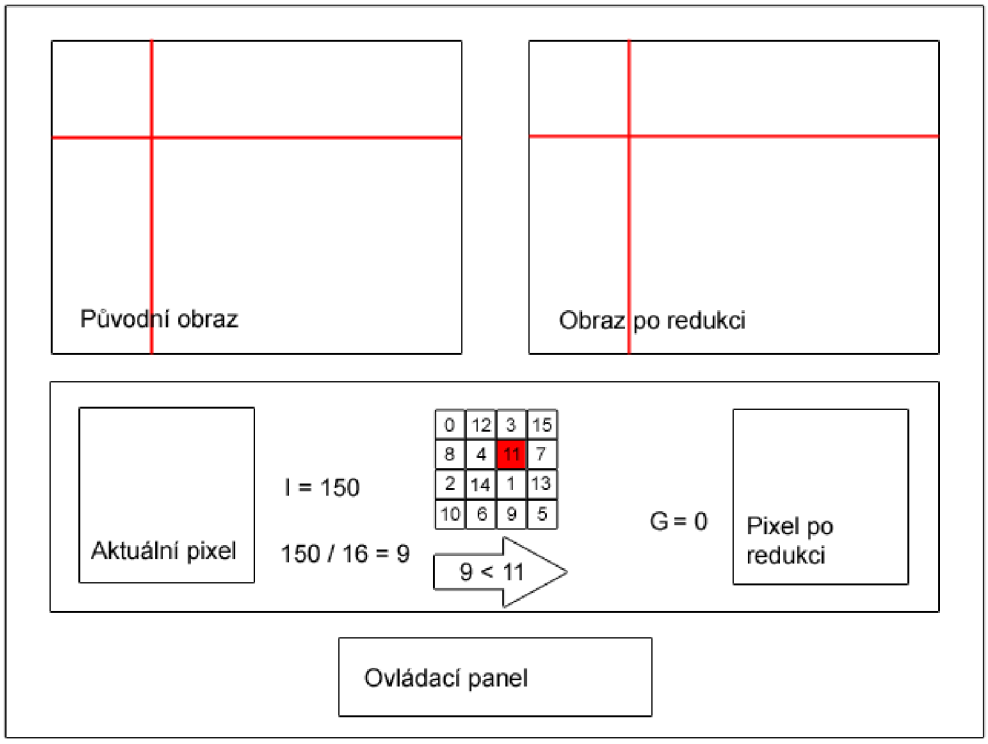
Obrázek 3.3: Náhodné rozptýlení

$$\begin{vmatrix} 0 & 12 & 3 & 15 \\ 8 & 4 & 11 & 7 \\ 2 & 14 & 1 & 13 \\ 10 & 6 & 9 & 5 \end{vmatrix}$$

Uživatel má možnost před začátkem redukce tuto matici dle libosti upravit (za podmínky, že zůstane zachována její konzistence). Podobně jako v předchozích případech bude zobrazen aktuální pixel ze vstupního obrazu (stupně šedi), jeho intenzita a její hodnota transformovaná na celočíselnou hodnotu od 0 do 15. Tato hodnota je poté porovnána s hodnotou uloženou v matici na příslušné pozici. V matici se zvýrazní hodnota, se kterou se aktuálně počítá. Stejně jako v předchozích případech, je výsledkem buď černý nebo bílý pixel, který se uloží do výstupního obrazu.

3.6 Distribuce chyby

Distribuce chyby není samostatné okno programu, nýbrž jen funkce, kterou si může uživatel zapnout u jakéhokoliv algoritmu a která způsobí jednak zobrazení příslušného panelu v okně, jednak bude daná metoda aplikována s ditribucí chyby. Panel má dvě části. V jedné se spočítá chyba vzniklá při redukci aktuálního pixelu (E). Ve druhé části okna se graficky zobrazuje princip distribuce chyby na okolní pixely. V tomto případě je použita Floyd-Steinbergova distribuce chyby, charakteristická následující maticí:



Obrázek 3.4: Maticové rozptýlení - dithering

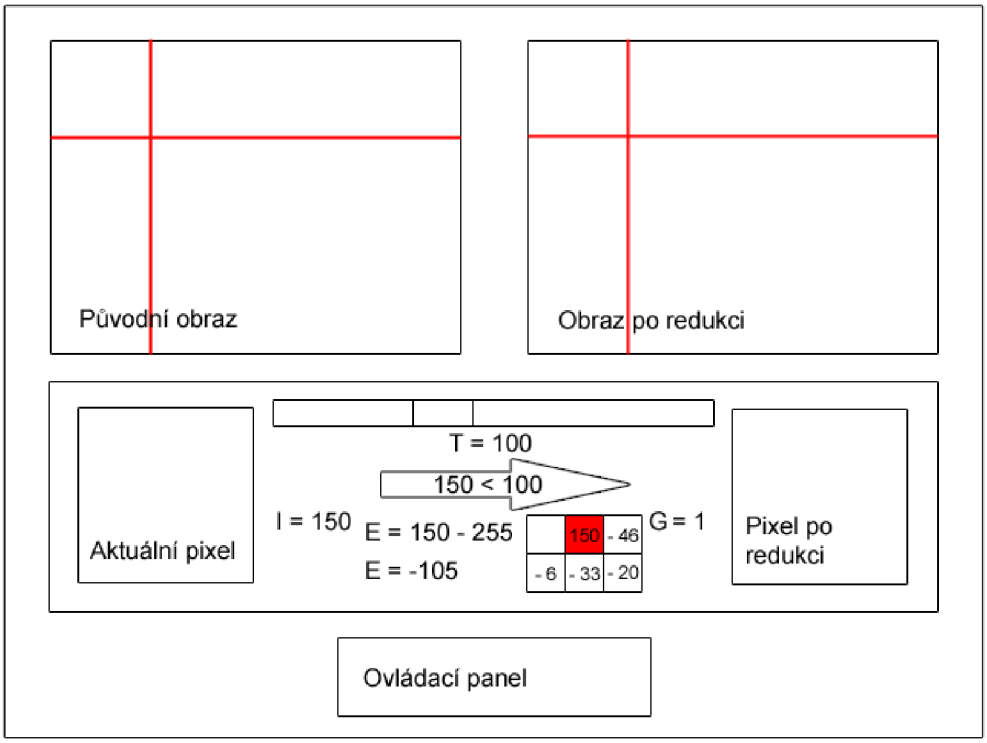
$$\begin{vmatrix} 0 & 0 & 0 \\ 0 & X & 7 \\ 3 & 5 & 1 \end{vmatrix}$$

Detailnější popis distribuce chyby je teoreticky popsán výše, nyní se budeme zabývat praktickou implemetací zejména distribuce chyby pro jednotlivé barevné složky. Jak víme z části 2.3.4, při distribuci chyby pouze z odstínů šedi vznikne jedna hodnota, která se posléze uloží do chybové matice. To je místo v paměti, které má stejný počet prvků jako je bodů v obrázku a kam se ukládají hodnoty rozdistribuované chyby pro daný pixel, které se pak berou v potaz při výpočtu intenzity daného bodu.

Ovšem problém nastává u barevné distribuce, kdy máme tři různé hodnoty. Za tímto účelem jsem použil vlastní metodu kódování této informace takovým způsobem, aby bylo možné použít stávající chybovou matici, a nemusel jsem vytvářet novou pro každou barevnou složku zvlášť. Výsledkem tohoto kódování je 27 bitové číslo, které se uloží do matice, jež v sobě obsahuje informace o chybě všech barevných složek. Prvních osm bitů zprava udává absolutní hodnotu chyby modré složky (0 až 255), na dalších osmi bitech je uložena absolutní hodnota chyby zelené složky, stejně tak je uložena i chyba červené složky. Zbývající tři bity jsou určeny pro příznak znaménka. Pokud je bit pro danou složku nastaven na 1, tak se její uložená hodnota uvažuje záporně, pokud na 0, tak kladně. Bity příznaků jsou zprava pro zelenou, modrou a červenou složku.

Například pokud chceme zakódovat následující hodnoty chyby:

$$R = 57, G = -174, B = 15$$



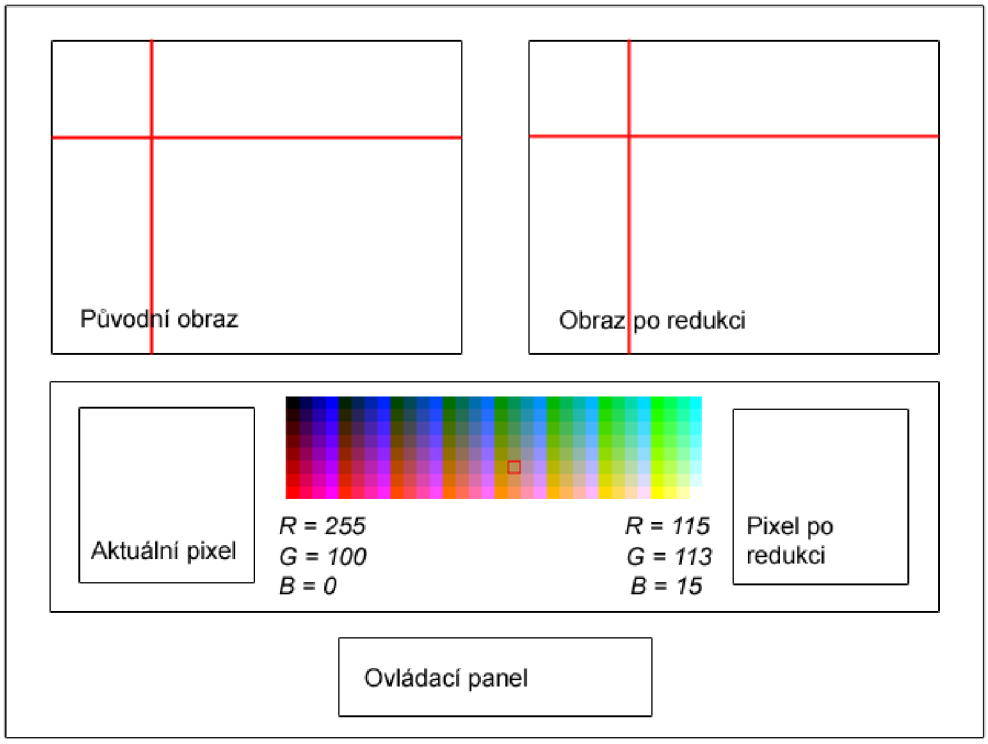
Obrázek 3.5: Distribuce chyby při prahování

Postupujeme tak, že si spočítáme absolutní hodnotu modré složky, což je v binárním tvaru 00001111, zleva k ní připojíme absolutní hodnotu chyby zelené složky 10101110 a totéž pro složku červenou 00111001. Takto nám vyjde číslo 00111001 10101110 00001111, k čemuž ještě přidáme informace o znaménkách, což je (v pořadí červená, zelená, modrá) 010. Výsledným zakódovaným číslem tedy je 010 00111001 10101110 00001111, v desítkovém zápisu 37334543. Toto číslo se uloží do chybové matice. Jsme schopni následně jednoduchým způsobem zjistit chybu kterékoliv složky.

3.7 Převod na paletu 3 - 3 - 2

V této části popíšu postup převodu obrázku na 256 odstínů palety 3 - 3 - 2. Tato paleta se používá tehdy, když chceme zredukovat barevnou informaci z důvodu úspory místa na 8 bitů. Z toho jsou první tři bity zleva použity pro uložení červené složky, další 3 bity pro uložení zelené složky a poslední dva pro složku modrou. Tato metoda ztratí podstatné množství informací o barvě, ovšem při jejím použití v kombinaci s barevnou distribucí chyby již vykazuje velmi zajímavé výsledky.

Nejprve se vytvoří samotná paleta a nainicializují se příslušné RGB barevné hodnoty. Ty se vypočítávají pomocí algoritmu, který určí podíl červené, zelené a modré složky v odstínu s indexem i z intervalu 0 až 255 následujícím způsobem:



Obrázek 3.6: Převod barevného obrázku na paletu 3 - 3 - 2

$$\begin{aligned} r &= (((i >> 5)) * 255)/7 \\ g &= (((i >> 2)\&7) * 255)/7 \\ b &= ((i\&3) * 255)/3 \end{aligned}$$

Jakmile máme sestavenou paletu barev, seřazenou podle indexu, tak můžeme přikročit k samotné redukci. Vezmeme si jednotlivé barevné složky vstupního pixelu a převedeme z osmibitové hodnoty na hodnotu tříbitovou pro červenou a zelenou složku, resp. hodnotu dvoubitovou pro složku modrou. Toto provedeme za pomocí následujícího vztahu.

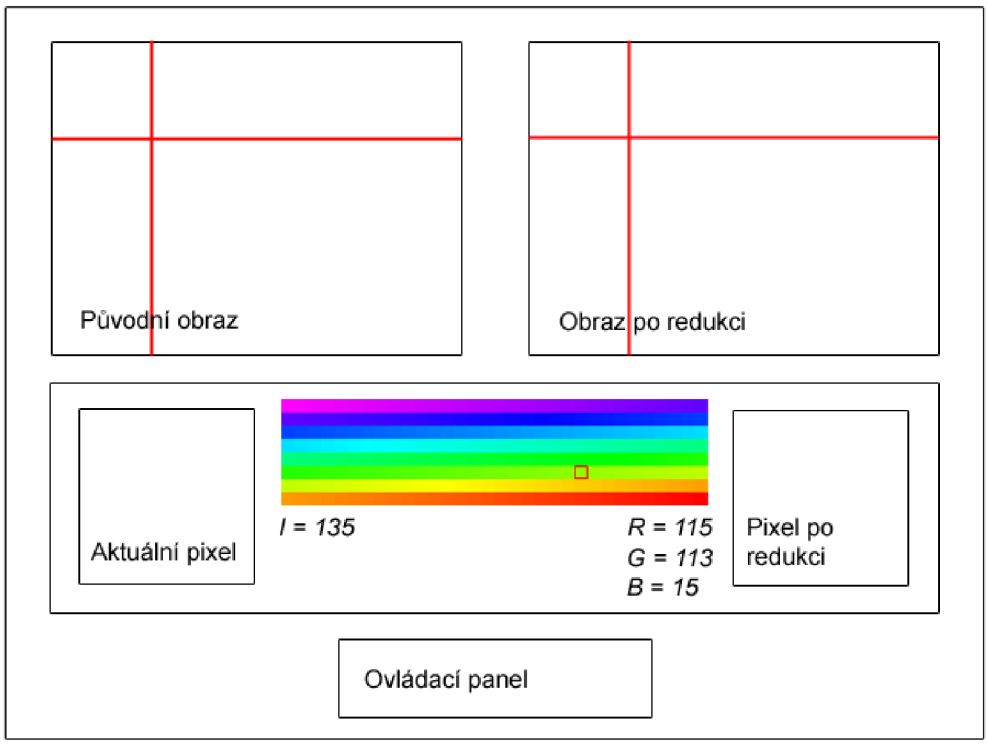
$$\begin{aligned} r_3 &= (r_8 * 7)/255 \\ g_3 &= (g_8 * 7)/255 \\ b_2 &= (b_8 * 3)/255 \end{aligned}$$

V této fázi již můžeme vypočítat index barvy v paletě příslušející barvě vstupního pixelu:

$$i = (r_3 << 5) + (g_3 << 2) + b_2$$

Podle tohoto vypočítanáho indexu zjistíme v poli barev palety, jaká RGB hodnota náleží danému odstínu v paletě 3 - 3 - 2, zvýrazníme ji ve vygenerované paletě a nastavíme tuto barvu u výstupního pixelu.

3.8 Převod na paletu false colors



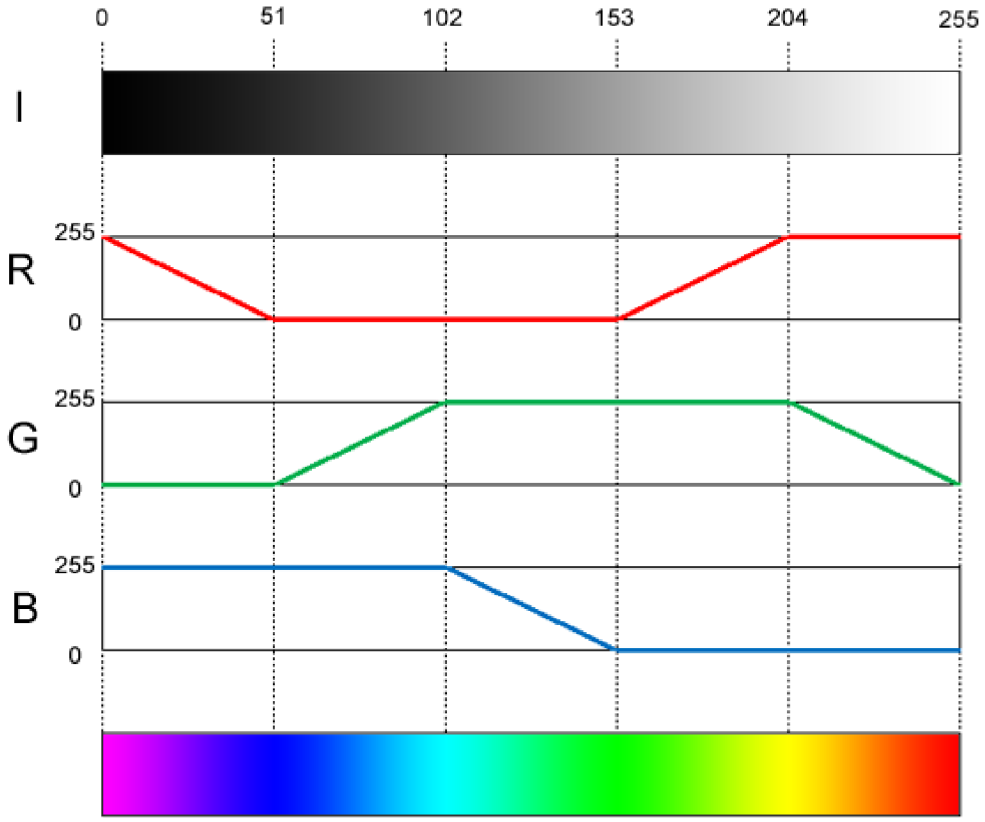
Obrázek 3.7: Převod barevného obrázku na paletu false colors

Převod na paletu false colors je v mnoha ohledech podobný jako v předcházejícím případě. Paleta false colors, kterou používám, má 256 odstínů rozdělených podle intenzity vstupního pixelu na barevné spektrum. Nejmenší hodnotě intenzity vstupního pixelu odpovídá fialová barva, nejvyšší hodnotě barva červená. Na obrázku 3.8 je znázorněn průběh jednotlivých barevných složek (červená, zelená, modrá) použitý při výpočtu výsledné barvy z palety. Na rozdíl od palety 3 - 3 - 2 není pro samotný průběh redukce nutné nejprve celou paletu hodnot vytvořit v paměti, poněvadž příslušnou barvu vypočítáme za běhu jen za pomoci vstupní intenzity.

V samotném programu je v rámci úspory výkonu tato paleta již vygenerována a v průběhu simulace se pouze vybírá barva podle intenzity. Stejně jako u palety 3 - 3 - 2 uživatel vidí celou paletu na hlavním panelu a také vidí, která z jejích barev byla právě vybrána.

3.9 Převod na paletu gradientu

Princip tohoto převodu je takový, že ze dvou vstupních barev se vytvoří přechod z jedné do druhé, a následně se tento přechod navzorkuje na 256 barev. Z této diskrétní barevné palety se pak vybírá výsledná barva příslušného pixelu v závislosti na jeho světelné intenzitě. Převod na 256 odstínů šedi je vlastně specializovaným případem barevného gradientu, kde vstupními barvami jsou černá a bílá.



Obrázek 3.8: paleta false colors

Uživatel má možnost výběru obou vstupních barev, na jejichž gradient se následně zredukuje vstupní bitmapa.

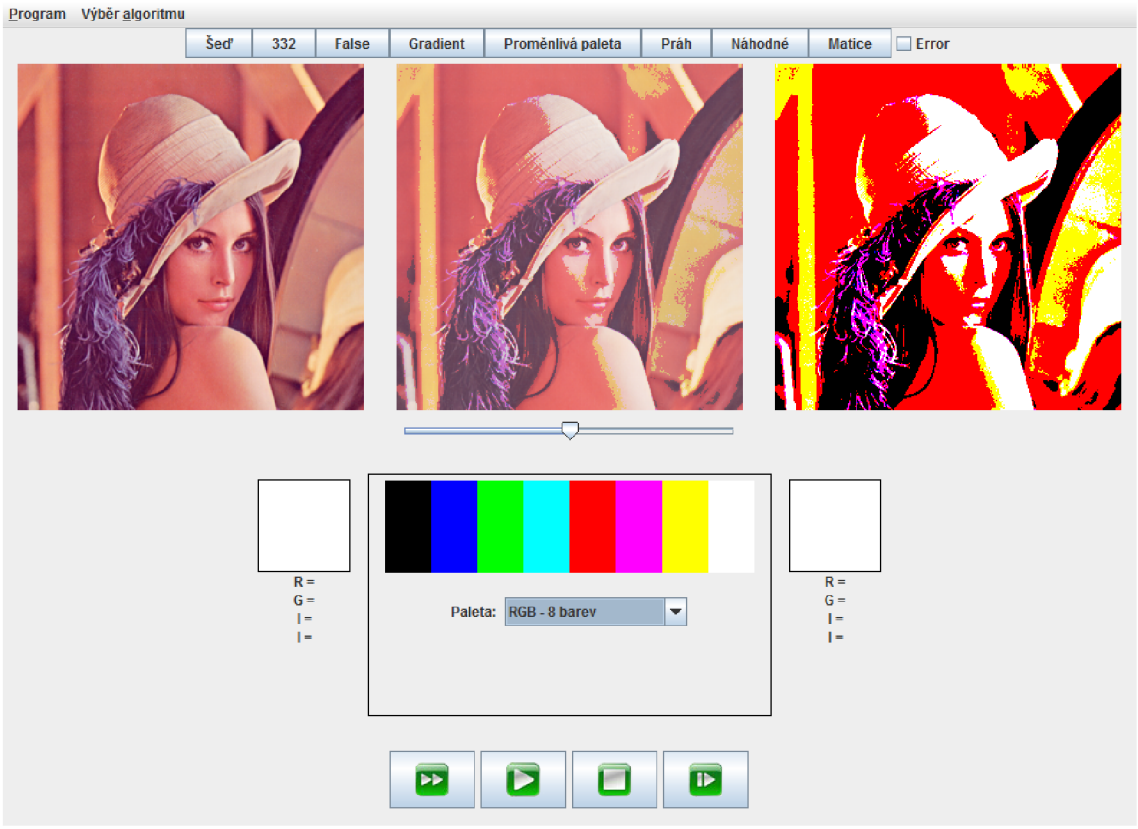
3.10 Proměnlivá barevná paleta

Poslední částí programu je sekce s možností převodu obrázku na předem zvolenou statickou barevnou paletu. Na výběr jsou následující palety:

- 2 odstíny šedi (1 bit) - černá a bílá (tato varianta odpovídá prahování s prahem 128)
- 4 odstíny šedi (2 bity)
- 16 odstínů šedi (4 bity)
- 256 odstínů šedi (8 bitů) - v tomto případě se jedná o klasický převod do stupňů šedi uvedený zde 3.1
- 8 barev (3 bity) - paleta používaná například v teletextu
- 64 barev (6 bitů) - paleta EGA (Enhanced Graphics Adapter)
- 256 barev (8 bitů) - paleta 3 - 3 - 2

Uživatel si může vybrat jednu z těchto palet a převést na ni vstupní obrázek.

3.11 Výsledná implementace programu



Obrázek 3.9: výsledné uživatelské rozhraní programu

Grafické uživatelské rozhraní programu prošlo několika změnami oproti původnímu návrhu. Na první pohled nejvýznamnější změnou je začlenění třetího panelu s obrázkem a posuvníkem. Tento panel slouží k přímému porovnání vstupního a výstupního obrázku za pomoci jejich prolínání[1]. Pokud posuneme posuvníkem úplně doleva, uvidíme zdrojový obrázek, vpravo naopak obrázek výsledný. Posuvník slouží pro plynulý přechod mezi nimi.

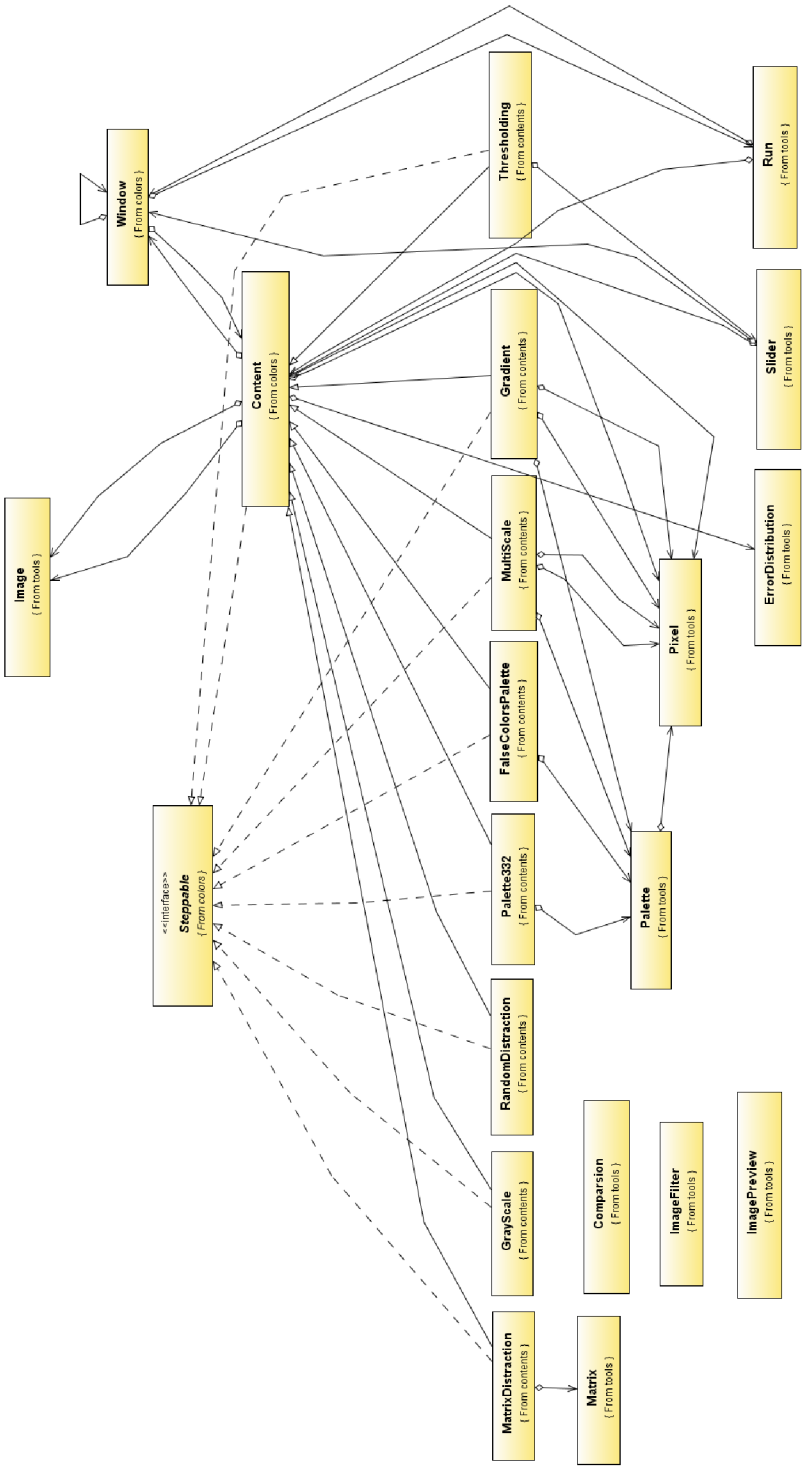
Další změnou je přidání horního panelu s tlačítky. Tato slouží k rychlému zvolení převodního algoritmu. To je z praktického hlediska při demonstraci programu ve výuce lepší než příslušný algoritmus hledat ve vysouvacím menu. Nicméně i tuto možnost uživatel má, neboť oba způsoby jsou ekvivalentní.

Právě z důvodu možnosti rychlejší demonstrace algoritmu bez toho, že bychom chtěli blíže zkoumat jeho princip, slouží volba Automatické vykonávání z menu Program. Tato možnost, ve výchozím stavu zapnutá, způsobuje to, že se zvolený algoritmus ihned vykoná a hned je vidět jeho výsledek. Při vypnutí tohoto přepínače by program čekal, až uživatel sám zapne animaci, nabídne mu možnost jejího pozastavení, krokování, či rychlého dokončení. V tomto případě může uživatel pozorovat, co se děje s každým pixelem vstupního obrázku, má možnost zapnout si za běhu distribuci chyby, či do jisté míry změnit parametry procesu.

[1]Pro bezproblémový chod prolínání doporučuji mít nainstalováno Java Runtime Environment (JRE) minimálně ve verzi 1.6.0_12

Uživatel má též možnost načíst si libovolný obrázek pro provedení barevné redukce a to buď volbou v menu, nebo přímo poklepáním na panel vstupního obrázku. Stejně tak má možnost uložit si soubor s výslednou redukovanou bitmapou ve formátu png poklepáním na panel s výstupním obrazem.

Co se ovládání týče, jsou zde čtyři hlavní tlačítka. Okamžité vykonání animace, spustit (pozastavit) animaci, zastavit animaci a vrátit do původního stavu, jakož i provést jeden krok animace. Pomocí nich je možno ovládat celý proces barevné redukce.



Obrázek 3.10: UML diagram programu

Kapitola 4

Výsledky

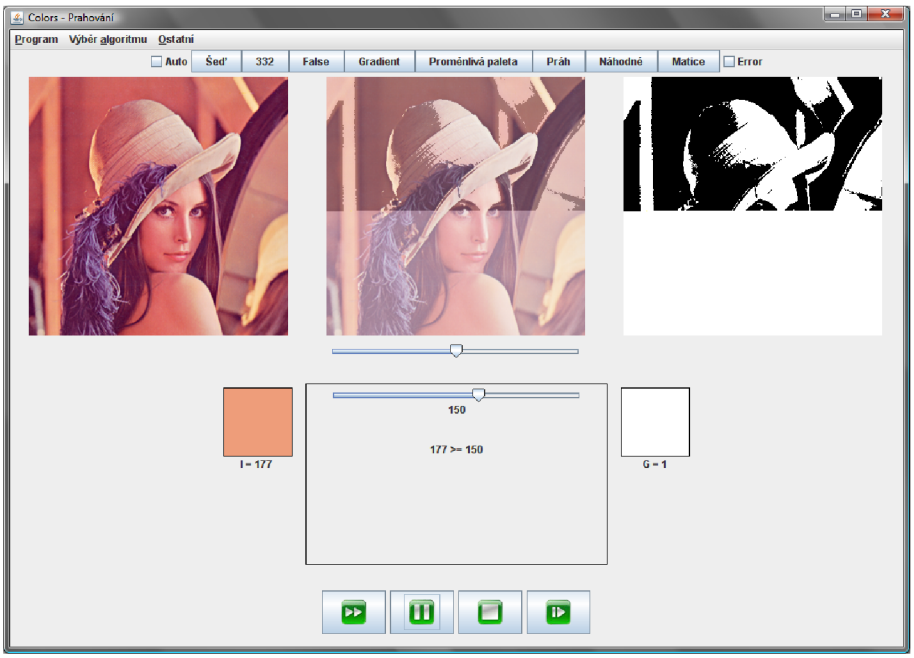
V této kapitole se zaměřím na popis výsledného programu, včetně popisu jeho funkcí a na to, jakým způsobem řeší konkrétní problematiku se zaměřením na její výukovou část. Jako příklad si uvedeme vykonávání metody prahování.

Obrázek 4.1: Základní okno programu

Ukážeme si postup při vypnutém automatickém vykonávání. V tomto případě vypadá okno programu v základním stavu jako na obrázku 4.1. Uživatelské rozhraní poskytuje následující důležité prvky:

1. Panel se vstupním obrázkem.
2. Panel s výsledným obrázkem.
3. Panel s posuvníkem ovládajícím prolínání vstupního a výsledného obrázku, což se zobrazuje na panelu uprostřed.

4. Lišta s tlačítky přepínajícímy jednotlivé obsahy, včetně checkboxu pro automatické vykonávání algoritmů a zapínání či vypínání distribuce chyby.
5. Tlačítka pro ovládání animace (rychlé vykonání, klasické spuštění či pauza, zastavení animace a nastavení výsledných hodnot, provedení jednoho kroku).
6. Aktuální pixel vstupního obrázku.
7. Pixel, který se uloží na odpovídající místo ve výsledném obrázku.
8. Hlavní panel se zobrazením informací o převodu a se specifickými ovládacími prvky (v tomto případě posuvník s výběrem prahové hodnoty)

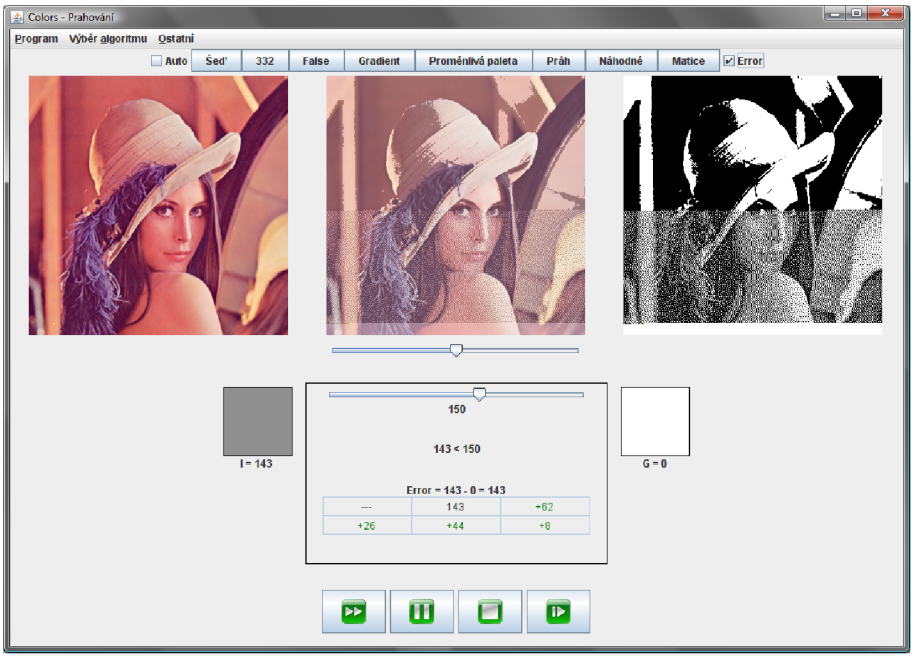


Obrázek 4.2: Program v průběhu animace

Jak vidíme, jsou všechny ovládací prvky nastaveny na výchozí hodnoty, výsledný obrázek je prázdný. Nyní si zvolíme hodnotu prahu v rozmezí od 0 do 255 (například 150), nastavíme tuto hodnotu na posuvníku a spustíme animaci tlačítkem „play“. Začne se postupně vykonávat algoritmus prahování, po jednom bodu. V průběhu animace se také na hlavním panelu budou zobrazovat informace o právě probíhajícím kroku, jak je to vidět na obrázku 4.2. Vidíme také prolínání obou obrázků na prostředním panelu.

V této chvíli zapneme distribuci chyby, abychom porovnali její efekt v rámci jednoho obrázku. Proto klikneme na checkbox Error na horní liště (případně stiskneme klávesovou zkratku Alt + D). Okamžitě se do procesu redukce začlení i rozdistribuovávání chyby, což nám indikuje také panel, který přibyl na spodní části hlavního panelu (4.3). Nově přidané informace nám ukazují, jak velká je hodnota chyby pro daný pixel a jaká její část se rozdistribuuje na okolní pixely.

Na obdobném principu pracují též ostatní obsahy programu, liší se jen množstvím zobrazených informací a ovládacími prvky na hlavním panelu, které jsou specifické pro příslušný algoritmus.



Obrázek 4.3: Program v průběhu animace po zapnutí distribuce chyby

4.1 Souhrnné informace o programu

Program byl napsán v jazyce Java, pro svůj běh tedy vyžaduje Java Runtime Environment, minimálně ve verzi 1.6.0_12 (kvůli opravené chybě s průhledností v některých operačních systémech). Pro korektní zobrazení prolínání je využita volně dostupná knihovna Swingx [6], obsažená ve verzi 0.9.5.

Cílem programu je zobrazit redukci barevné palety ze zdrojového na cílový obrázek a to jak přímo, tak s možností krokování postupu. V tom případě jsou zobrazeny podrobnější informace o převodu.

Aplikace umožňuje vizualizovat následující algoritmy (jejich screenshoty jsou na obrázku 4.4):

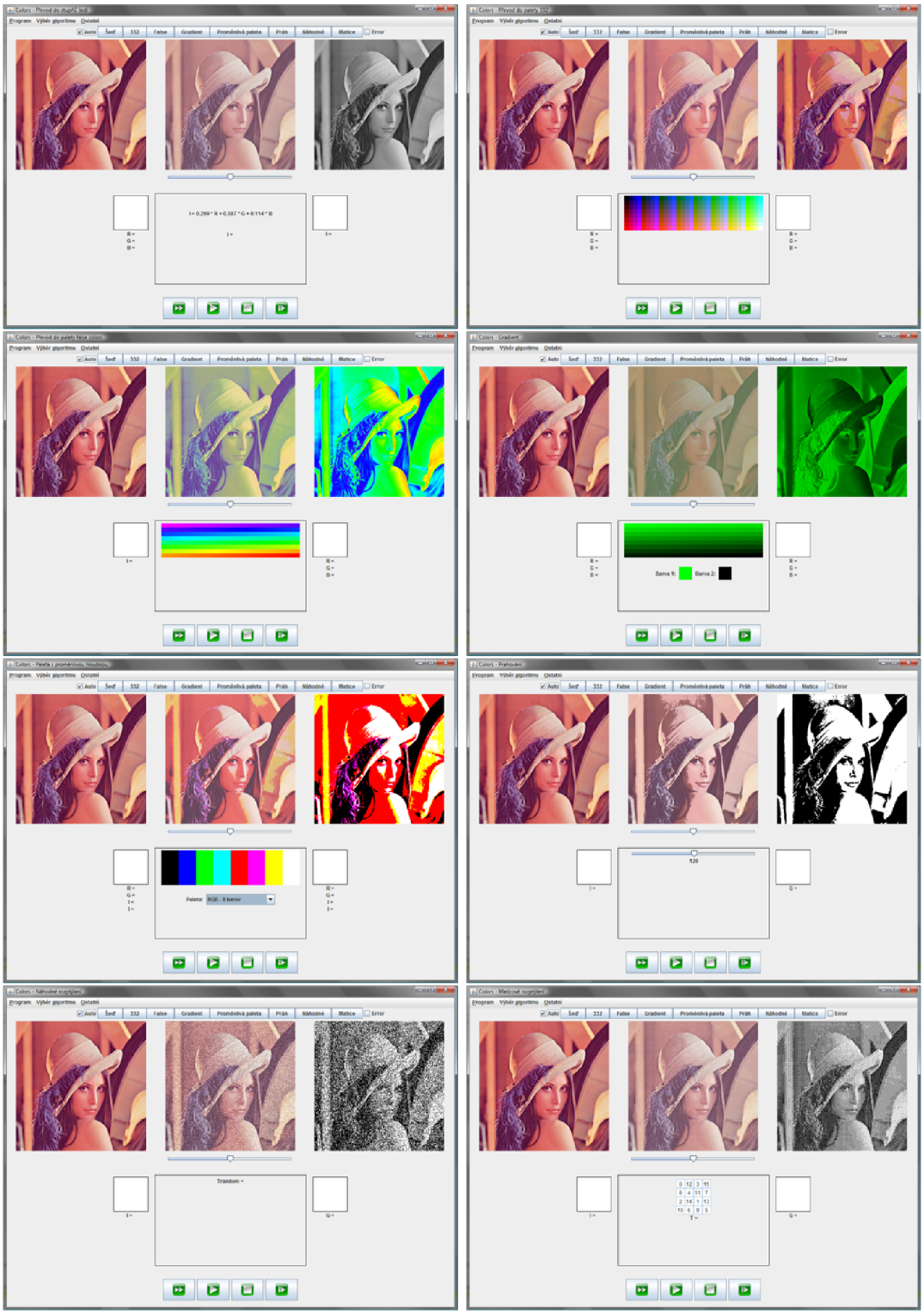
- Převod do 256 stupňů šedi.
- Převod na barevnou paletu 3 - 3 - 2.
- Převod na paletu false colors.
- Převod na paletu gradientu (přechod z jedné libovolně zvolené barvy do druhé).
- Redukce barevného prostoru na následující palety:
  - Monochromatické palety
    - 2 stupně šedi
    - 4 stupně šedi
    - 16 stupňů šedi
    - 256 stupňů šedi

  - Barevné palety
    * 8 barev
    * 64 barev
    * 256 barev (paleta 3 - 3 - 2)

- Prahování s možností změny prahu
- Náhodné rozptýlení
- Maticové rozptýlení metodou ditheringu s možností změny distribuční matice.

Na všechny tyto postupy lze aplikovat distribuci chyby. Dále je zde možnost nahrávat si své vlatní obrázky (pro tyto účely jsou přibaleny vhodné ukázky). V menu Program můžeme pro lepší demonstraci změnit velikost obrázku (na výběr máme hodnoty 200, 300, 400 a 500 pixelů). Výsledek procesu si můžeme pro další zkoumání uložit ve formátu png.

Program umožňuje dva režimy provozu. V režimu „Automatické vykonávání“, jež je ve výchozím stavu zapnutý, se zvolený algoritmus okamžitě vykoná a jakákoliv změna parametrů (změna prahu, změna barvy u gradientu) má za následek okamžité provedení celého procesu a zobrazení výsledku. Pokud tuto možnost uživatel vypne, musí spustit animaci ručně (má na výběr rychlé vykonání, klasické spuštění a krokování).



Obrázek 4.4: Snímky obrazovky pro jednotlivé obsahy

Kapitola 5

Závěr

Cílem tohoto programu bylo nezaměřit se jen na prosté předvedení výsledku daného algoritmu, ale především poodhalit uživateli princip, za pomoci kterého se k výsledku došlo. Prostředkem ke splnění tohoto cílu bylo zejména zjednodušení uživatelského rozhraní s důrazem na prvky s informativním charakterem (panely s podrobnostmi o průběhu animace). Byly použity všechny základní a v praxi nejčastěji používané algoritmy redukce barevného prostoru. Díky možnosti nahrát si jakýkoliv obrázek pro úpravu má uživatel možnost porovnat různé efekty dané metody na odlišné obrázky. Za tímto účelem je jich několik připraveno pro snadnější demonstraci.

Aplikace je navržena s tím úmyslem, aby bylo možné i její budoucí rozšíření. Bez větších problémů je možno přidat jakýkoliv algoritmus založený na podobném principu, stejně tak jako je možné rozšířit množství použitých redukčních palet. Dalšími kandidáty na rozšíření funkčnosti programu mohou být například adaptivní prahování, paleta dynamicky generovaná v závislosti na histogramech vstupního obrázku a další.

Tento program je možno využít například při výuce této problematiky v rámci předmětu Základy počítačové grafiky (IZG), případně i na středních školách v předmětech zabývajících se touto tematikou.

Literatura

[1] Archiv radarových snímků ČHMÚ. [online], Naposledy navštíveno květen 2009.
URL http://www.chmu.cz/meteo/rad/index.html

[2] Dithering. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/Dithering

[3] Edge detection. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/Edge_detection

[4] Enhanced Graphics Adapter. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/Enhanced_Graphics_Adapter

[5] Halftoning. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/Halftone

[6] Java Swing component extensions. [online], Naposledy navštíveno květen 2009.
URL https://swingx.dev.java.net/

[7] Linear congruential generator. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/Linear_congruential_generator

[8] List of monochrome and RGB palettes. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/List_of_monochrome_and_RGB_palettes

[9] Ordered dithering. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/Ordered_dithering

[10] Thresholding. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/Thresholding_(image_processing)

[11] TrueColor palette. [online], Naposledy navštíveno květen 2009.
URL http://en.wikipedia.org/wiki/Truecolor

[12] Jiří Žára, J. S. P. F., Bedřich Beneš: *Moderní počítačová grafika*. Computer press, 2004, iSBN 80-251-0454-0.

[13] Kršek, P.: Základy počítačové grafiky, studijní opora k předmětu IZG, Fakulta informačních technologií VUT v Brně.