Západočeská univerzita v Plzni
Fakulta aplikovaných věd
Katedra informatiky a výpočetní techniky

Bakalářská práce

Systém pro barevné korekce digitálních fotografií

Plzeň, 2009 Věra Skorkovská

Zadání

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracovala samostatně a výhradně s použitím citovaných pramenů.

V Plzni dne ....

Digital image color correction system

Color correction of digital images is a very important part of prepress but it can be useful for a non-professional photographer as well. There are many programs suitable for color correction, each of them has its pros and cons. The professional ones are quite expensive for a common user, free software usually lacks some of the fundamental tools, such as the Curves or the Info palette. Therefore a request arose to create a portable color correction system, which would be distributed as open source and which would offer the tools mentioned above.

This thesis will look closely into the basic tools for color correction and afterwards use the gained experience to implement the desired application, which will be thoroughly documented for the needs of future expansions.

Obsah

1. Úvod

Barevné korekce jsou v současné době neoddělitelně spjaty s vytvářením jakékoli fotografie pro komerční použití. Žádný z obrázků, které denně vídáte v časopisech, reklamních letácích či na plakátech, se k Vám nedostane, dokud neprojde velmi pečlivou předtiskovou úpravou, jejíž nedílnou součástí jsou barevné korekce.

Barevné korekce však mají smysl i pro amatérského fotografa, který se chce přátelům pochlubit krásnými fotkami z dovolené. Ne vždy jsou fotografie pořízeny za příznivých podmínek, což vede k jejich špatnému kontrastu či barevnosti. V takovou chvíli přicházejí na řadu barevné korekce, s jejichž pomocí lze většinu těchto problémů odstranit, či alespoň minimalizovat.

Existuje velká řada programů, ve kterých je možné provádět barevné korekce, každý z nich má své klady i zápory. Většina těch kvalitnějších je pro běžného uživatele nevhodná kvůli vysoké ceně, software zdarma obvykle postrádá některé ze stěžejních funkcí (jedná se především o nástroj Křivky a paletu Informace pro přesné odečítání hodnot pixelu). Proto vznikl požadavek na vytvoření přenositelné aplikace pro barevné korekce, která bude šířena jako *open source*[1] a která bude poskytovat potřebné nástroje.

V této práci nejprve prozkoumám nejčastěji používané nástroje pro barevné korekce, zmíním základní barevné prostory a blíže popíši nejznámější software, zaměřující se na barevné korekce.

V druhé fázi implementuji přenositelnou aplikaci pro úpravu digitálních fotografií, součástí implementace bude některý z nejdůležitějších nástrojů pro barevné korekce. K programu vytvořím důslednou dokumentaci datových struktur, infrastruktury a zdrojového textu, díky tomu bude možné práci dále rozšiřovat podle aktuálních potřeb.

Implementovanou aplikaci následně otestuji a na závěr shrnu výsledky mé práce a získané poznatky.

[1] Software s otevřeným zdrojovým kódem, jehož licence umožňuje uživatelům zdrojový kód využívat, prohlížet a upravovat.

2. Teoretická část

2.1. Barevné korekce

I přestože se výrobci digitálních fotoaparátů snaží produkovat stále lepší a lepší přístroje, které budou schopné vytvořit kvalitní fotografii i za nepříznivých vnějších podmínek, potřeba barevných korekcí je (a pravděpodobně vždy bude) stále velká. Mnohé fotografie jsou samostatně použitelné přímo ve stavu, jak je vytvoří digitální fotoaparát. Pokud je však chceme použít současně s dalšími, například v prezentaci, je nutné všechny fotografie barevně sladit, aby působily jednotným dojmem.

Dalším důvodem pro použití barevných korekcí může být potřeba upravení kontrastu snímku při prohlížení na monitoru, projektoru či papíře, žádné z těchto zařízení neposkytuje tak velký kontrast, jaký dokáže zachytit fotoaparát.

Nejčastějším přístupem k barevným korekcím je ale snaha o co nejvěrnější zachycení obrazu, jak by ho viděl člověk nacházející se na pozici fotoaparátu. Lidský vizuální systém vyhodnocuje barvy v kontextu, ne absolutně. Přizpůsobuje se barvě *ambientního*[2] světla a vnímá jej jako neutrální. Předmět, na který se člověk soustředí, navíc získává na kontrastu, vše ostatní se jeví méně kontrastní. Existuje mnoho dalších zvláštností lidského vidění, které způsobují, že pořízený snímek často vypadá výrazně jinak, než jak si jej pamatuje člověk, který ho fotil. Barevné korekce nám poskytují mocný nástroj k dosažení požadovaného efektu – napodobení lidského vnímání světa.

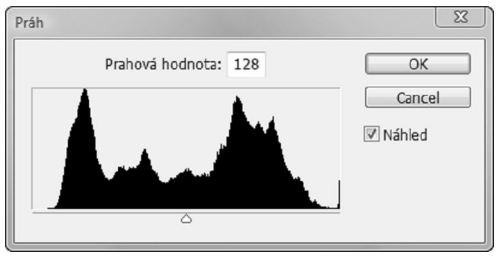
2.2. Základní nástroje pro korekci barev

Abychom byli schopni sami provádět barevné korekce obrázků, musíme nejprve porozumět základům fungování jednotlivých nástrojů. V následující části práce představím nejpoužívanější nástroje, které jsou implementovány ve většině programů určených pro barevné korekce. Vzhled nástrojů se může aplikaci od aplikace mírně lišit, princip však zůstává pořád stejný. Pro ilustraci budu používat ukázky nástrojů z programu *Adobe Photoshop*.

[2] Světlo obklopující objekt či scénu, u nějž nedokážeme určit, z kterého zdroje pochází.

2.2.1. Práh

Práh (obr. 2.1) patří mezi nejjednodušší nástroje pro korekci barev. Umožňuje nastavit prahovou hodnotu jasu, podle níž se přepočtou jednotlivé pixely[3] obrázku – pixely s jasem vyšším než práh budou ve výsledku bílé, pixely s nižším jasem budou černé. *Práh* lze použít pro tvorbu černobílých obrázků, častěji je však používán jen jako pomocný nástroj. Posouváním hodnoty prahu lze jednoduše zjistit, které části obrázku jsou nejtmavší a které naopak nejsvětlejší. Tato informace má při barevných korekcích často zásadní význam pro správné nastavení světel a stínů[4].



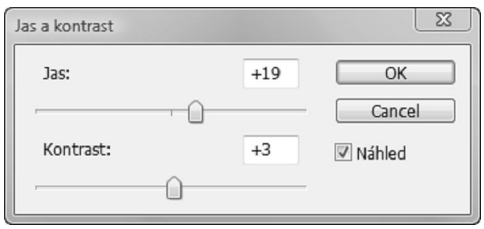
Obr. 2.1: Nástroj Práh

2.2.2. Jas – kontrast

O něco složitějším nástrojem je *Jas – kontrast* (obr. 2.2). Jak název napovídá, slouží k nastavení jasu a kontrastu obrázku.

Změnou jasu ovlivňujeme všechny barevné kanály pixelu ve stejné míře. Při zvýšení jasu je ke každému kanálu přičtena zadaná hodnota, při snížení jasu je tato hodnota odečtena. Kontrast ovlivňuje způsob, jakým jsou interpretována světla a stíny v obrázku. Zvýšením kontrastu posuneme světla blíže k bílé barvě a stíny blíže k černé, snížení kontrastu má opačný účinek.

Používat tento nástroj však není příliš vhodné. Na všechny obrazové body aplikuje stejnou korekci, což může vést ke ztrátě detailu [2].



Obr. 2.2: Nástroj Jas a kontrast

[3] Pixel, či obrazový bod, je nejmenší jednotkou digitální rastrové grafiky. Představuje jeden bod obrázku.

[4] Světlem (highlight) bývá při barevných korekcích nazývána nejsvětlejší oblast fotografie, stínem (shadow) naopak nejtmavší oblast.

2.2.3. Úrovně

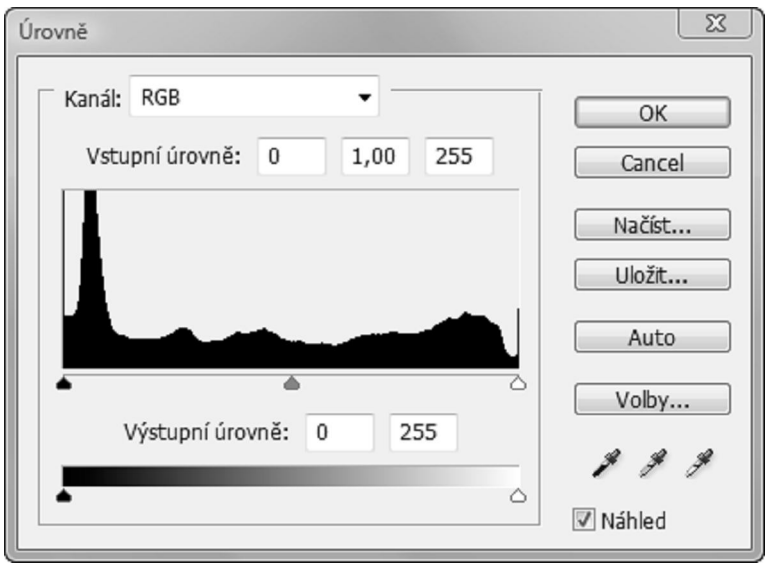
Komplexnějším nástrojem pro barevnou korekci jsou *Úrovně* (viz obr. 2.3).

Úrovně umožňují nastavení černého bodu, bílého bodu, středních tónů a změnu kontrastu obrázku, tedy poměru nejsvětlejšího ku nejtmavšímu bodu obrázku. Nástroj zobrazuje histogram jednotlivých úrovní intenzity, jehož vyhodnocení pomůže s určením problémů upravovaného obrázku. Pokud histogram není roztažený přes celou šířku, obrázek nevyužívá celý rozsah tónů a ztrácí tak kontrast. Tento problém lze ve většině případů odstranit posunem černého a bílého bodu, některé fotografie, jako například snímky krajiny v mlze, naopak plný histogram mít nikdy nemohou.

Posun černého bodu na určitou hodnotu způsobí, že všechny pixely s touto nebo nižší hodnotou budou změněny na černé, obdobně posunem bílého bodu se změní pixely se stejnou nebo vyšší hodnotou na bílé. Změna nastavení středních tónů ovlivní hodnoty intenzity ve středním rozsahu šedých tónů, aniž by výrazně ovlivnila světla a stíny. Snížením hodnoty výsledný obrázek zesvětlíme, naopak zvýšením obrázek ztmavíme [3].

K ovlivnění kontrastu slouží výstupní úrovně. Posunem krajních bodů měníme černá, resp. bílá místa na požadovanou úroveň šedé [1].

Úrovně umožňují upravovat nejen jasový kanál, ale i jednotlivé barevné kanály. Při úpravě konkrétního kanálu je princip fungování nástroje shodný, ale změny se aplikují pouze na vybraný kanál.

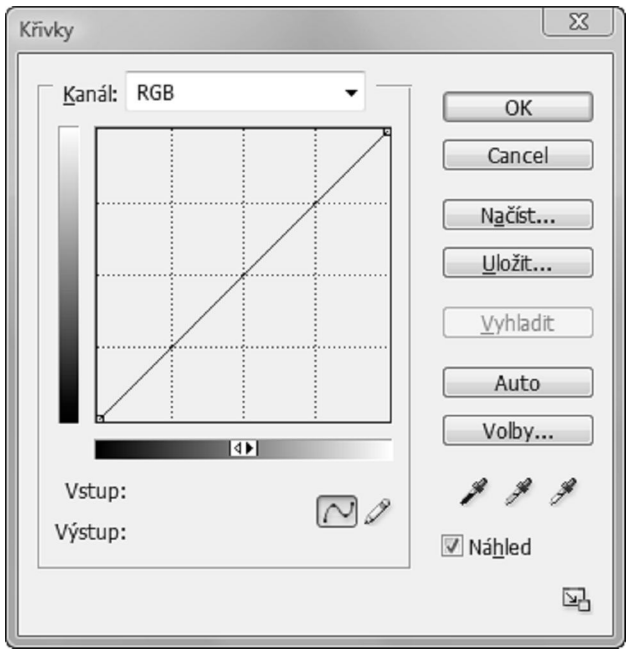


Obr. 2.3: Nástroj Úrovně

2.2.4. Křivky

Nástrojem, který nám nejlépe dovoluje kontrolovat změny prováděné v obrázku, jsou *Křivky*. Na rozdíl od *Úrovní*, které umožňují měnit pouze nastavení černého a bílého bodu a středních tónů, *Křivky* mohou provádět barevnou korekci libovolného bodu.

Horizontální osa *Křivek* reprezentuje současné hodnoty, na vertikální ose jsou vyneseny hodnoty po provedení barevné korekce. Po vyvolání nástroje se dá křivka popsat lineární funkcí $f(x) = x$, výstupní hodnota je tedy rovna hodnotě vstupní (viz obr. 2.4). Tato křivka má pouze dva body – černý bod [0, 0] a bílý bod [255, 255]. Horizontálním posunem těchto bodů můžeme upravovat využívaný rozsah tónů. V naprosté většině případů je naším cílem obrázek, který využívá celý rozsah, proto je rozumné nastavit černý bod na nejtmavší místo a bílý bod na nejsvětlejší místo obrázku.



Obr. 2.4: Nástroj Křivky

Přidáváním nových bodů do křivky a jejich posunem nahoru a dolů měníme tvar převodní funkce. Například posunem bodu z [50, 50] do [50, 60] říkáme, že všechny pixely, které mají hodnotu 50, mají mít ve výsledku hodnotu 60. Dojde tedy k jejich zesvětlení. Obdobně posunutím bodu dolů dojde k ztmavení výsledného obrázku.

Změnou tvaru křivky je možné měnit lokální kontrast částí obrazu, tzn. poměr nejsvětlejšího a nejtmavšího bodu oblasti. Výchozí křivka $f(x) = x$ stoupá pod úhlem 45°, přidáním nových bodů a jejich vertikálním posunem vzniknou v některých částech

křivky prudší oblasti a v jiných oblasti plošší. V částech obrázku spadajících do prudší oblasti křivky se původní rozsah tónů přemapuje na rozsah větší, obrázek v těchto místech získá větší lokální kontrast. Oblasti s jasy, pro které je křivka plošší, naopak kontrast ztratí a budou ve výsledném obrázku vypadat hůře. Proto je nutné pečlivě zvážit, která část obrázku je nejdůležitější a kde je vhodné zvětšit kontrast, a která část je nepodstatná a snížení kontrastu příliš neublíží.

Křivky je možné aplikovat na kompozitní kanál (změny budou ovlivňovat všechny kanály) i pouze na jeden konkrétní barevný kanál. Je vhodnější upravovat každý kanál zvlášť, tímto postupem je možné dosáhnout lepších výsledků. [4]

Hodnoty mezi zadanými body křivky se mohou interpolovat různými způsoby. Nejjednodušším je spojení bodů lineárními úseky, toto řešení se však v profesionálních nástrojích často nepoužívá kvůli efektu Machových proužků[5]. Jiným řešením je vytvoření interpolační křivky pomocí *Lagrangeovy interpolace*. Pro *n* vstupních bodů dostáváme polynom řádu *n-1*. Výsledná křivka může v důsledku vysokého řádu výrazně kmitat a při přidání nového bodu je nutné přepočítat celý polynom, z těchto důvodů je ještě méně vhodná než lineární interpolace.

Nejlepším a nejpoužívanějším řešením je použití *kubické spline funkce*, tedy křivky, která je po částech tvořena polynomy třetího stupně. Její výhodou je nízký řád polynomů, který omezuje kmitání funkce, a také to, že posun řídícího bodu změní tvar křivky pouze v omezeném okolí. Nutnost přepočítávání polynomu však zůstává, pokud vyžadujeme zachování C^2 spojitosti[6].

Postup výpočtu zmiňovaných interpolačních křivek je možné najít v [5].

2.2.5. Odstín – sytost

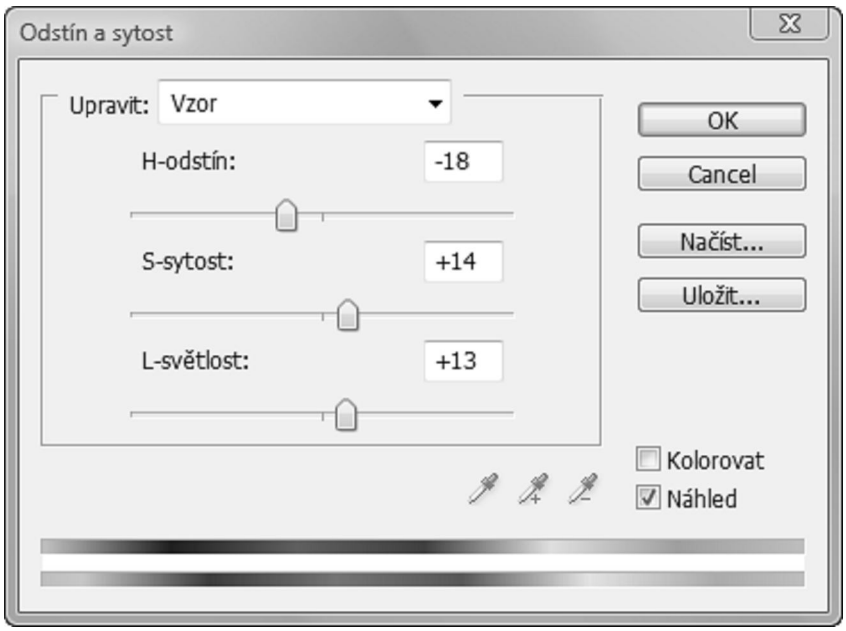
Nástroj *Odstín – sytost* (viz obr. 2.5) slouží ke změně odstínu, sytosti a světlosti v obrázku. Lze jej použít pro výraznou změnu barvy obrázku.

Změnou odstínu měníme barvy v obrázku. Sytost udává čistotu barvy – určuje, jak moc se výsledná barva odlišuje od neutrální barvy. Snížením sytosti na minimum získáme černobílý obrázek. Nastavení světlosti ovlivňuje jas výsledného obrázku.

Tento nástroj je možné používat pro změnu všech barev najednou nebo pro aplikaci změn pouze na konkrétní rozsah barevných tónů.[1]

[5] Lidské oko je citlivé na hranice změny jasu. Na hranici světlé a tmavé oblasti se světlá barva jeví ještě světlejší a tmavá ještě tmavší.

[6] Křivka je C^2 spojitá, pokud má spojité derivace až do druhého řádu.

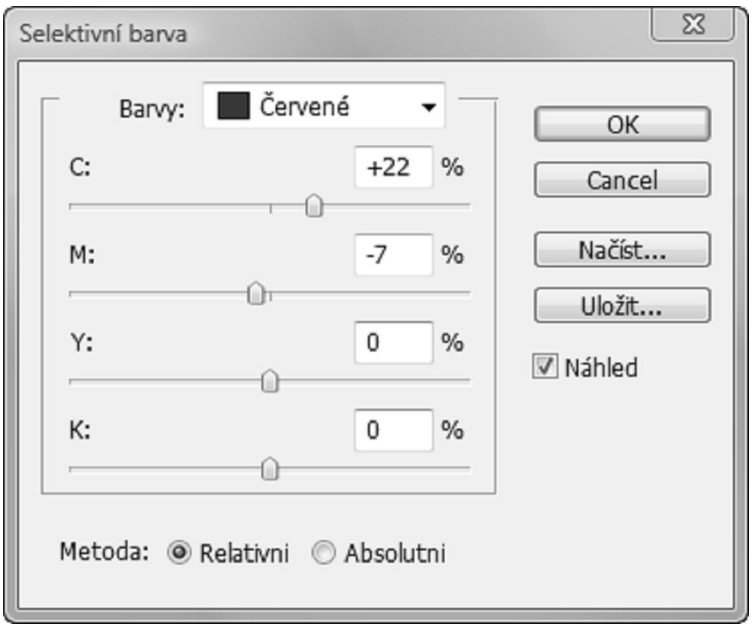


Obr. 2.5: Nástroj Odstín a sytost

2.2.6. Selektivní barva

Nástroj *Selektivní barva* (viz obr. 2.6) je vhodný pro jemné korekce, například doladění odstínů nebo ztmavení bílých míst.[6]

Nejprve je nutné zvolit barvu, kterou chceme měnit. Poté už můžeme posuvem jezdců směrem k základním barvám ovlivňovat vybrané barevné tóny.[7]

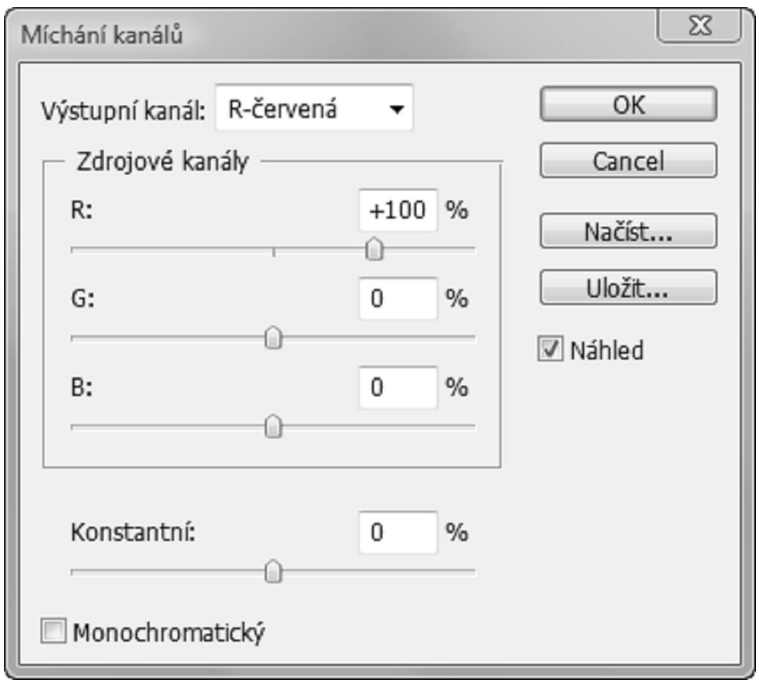


Obr. 2.6: Nástroj Selektivní barva

2.2.7. Míchání kanálů

Nástroj *Míchání kanálů* (obr. 2.7) pracuje s jednotlivými barevnými kanály obrázku. Jak název napovídá, dovoluje mezi sebou kanály libovolně kombinovat.

Po výběru výstupního barevného kanálu bude odpovídající barevná složka nastavena na 100%, ostatní složky budou mít nulovou hodnotu. Posunem jezdců zdrojových kanálů se do výstupního kanálu přidá či ubere dané množství světla v závislosti na obsahu jednotlivých kanálů a poloze jezdce. Výsledná hodnota barevného kanálu se vypočte podle vztahu $X = r * R + g * G + b * B$, kde X je zvolený výstupní kanál, r, g, b jsou hodnoty nastavené jezdci a R, G, B představují obsah kanálů původního obrázku. Součet hodnot r, g a b by měl dávat 100%.



Obr. 2.7: Nástroj Míchání kanálů

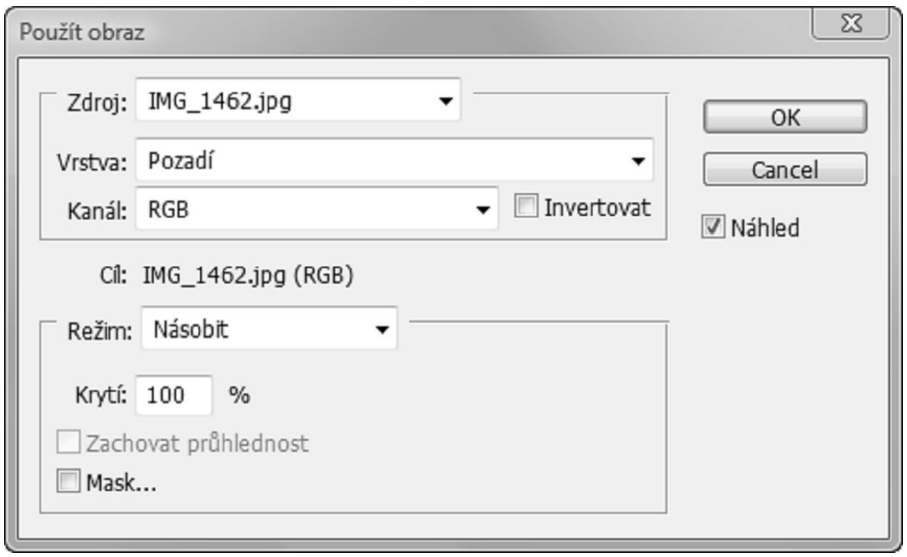
Míchání kanálů lze využít při vytváření kvalitních černobílých obrázků. Pouhým převedením obrazu na stupně šedi sice získáme černobílý obrázek, ale výsledek nemusí být příliš dobré kvality. Některé barvy, které se zdají být naprosto odlišné, mohou mít podobnou intenzitu jasu, a při převodu z nich vznikne téměř stejná šedá. Výsledný obrázek je pak velice nevýrazný. *Míchání kanálů* umožňuje určit, který z kanálů se má do výsledku promítnout výrazněji a který méně. Postupným prozkoumáním barevných kanálů zjistíme, který kanál nejlépe vystihuje naší představu o výsledném obrazu a naopak který kanál výsledný dojem nejvíce kazí. Na základě těchto poznatků můžeme

upravit intenzitu jednotlivých kanálů, abychom dosáhli požadovaného efektu. Případná změna barevnosti zdrojového obrázku je v tuto chvíli nepodstatná, při převodu na stupně šedi bude záležet pouze na jasu jednotlivých pixelů. [4]

Nástroj lze využít i pro korekci barevného obrazu, v takovém případě je ale nutné dávat větší pozor na nežádoucí změny barevnosti.

2.2.8. Použít obraz

Nástroj *Použít obraz* umožňuje zkombinovat vrstvu a kanál z jednoho obrázku (zdrojového) s vrstvou a kanálem jiného obrázku (cílového). Oba obrázky musí mít stejné rozměry (v pixelech), jinak je nástroj nedovolí použít. Před vyvoláním nástroje je nutné oba obrázky otevřít a zvolit, který z nich bude cílový. Poté už můžeme otevřít *Použít obraz* (obr. 2.8) a začít s úpravou.



Obr. 2.8: Nástroj Použít obraz

Nejprve zvolíme zdrojový obrázek, konkrétní vrstvu, kterou chceme použít, a barevný kanál, jež budeme aplikovat na cílový obrázek. Parametry cílového obrázku nelze měnit v tomto nástroji, automaticky se nastaví podle toho, co bylo označeno v době, kdy jsme dialog vyvolali. Nyní zbývá nastavit *režim prolnutí* a hodnotu *krytí*. *Režim prolnutí* určuje, jakým způsobem se bude zdrojový obrázek kombinovat s cílovým, zatímco hodnota *krytí* umožňuje ovlivnit sílu výsledného efektu. [8]

2.3. Světlo a barevné systémy

Světlem nazýváme viditelnou část elektromagnetického záření, tj. záření o vlnové délce asi 380 až 720 nm. Různé vlnové délky vnímáme jako různé barvy světla. Světla obsahující jedinou vlnovou délku (monochromatická) se v přírodě téměř nevyskytují, lidské oko je ale schopné interpretovat směs vlnových délek také jako barvu, některé barvy dokonce vůbec není možné vyjádřit jedinou vlnovou délkou.

Při popisu barvy bychom tedy teoreticky museli zaznamenat pro každý pixel intenzitu světla každé vlnové délky viditelného spektra, což by bylo velice paměťově náročné, nehledě na to, že výroba zařízení, které by bylo schopné zreprodukovat pro každý pixel přesnou směs vlnových délek dané intenzity, je pravděpodobně nemožná. Experimentálně bylo zjištěno, že všechny barvy lze namíchat z pouhých třech základních, a to červené, zelené a modré.

Bylo zavedeno mnoho barevných systémů, z nichž některé jsou silně svázány s určitou technologií, jiné jsou určené například pro snadný výběr barev. V následujícím textu popíšu ty nejdůležitější.

2.3.1. RGB

Aditivní systém *RGB* se používá u zařízení, která vyzařují světlo, například u barevných monitorů. Výsledná barva je tvořena součtem červené (R – red), zelené (G – green) a modré (B – blue) složky. Pro správné zobrazení barvy je nutné definovat spektra jednotlivých složek. Tyto hodnoty se mohou u různých zařízení výrazně lišit, což by vedlo k tomu, že by jeden obrázek měl na každém zařízení jinou barvu. Jako řešení tohoto problému byly zavedeny různé *RGB* standardy lišící se definicí primárních světel.

Nejčastěji používaným *RGB* barevným prostorem je *sRGB* (obr. 2.9a), který se používá ve většině monitorů a amatérských digitálních fotoaparátů. Tento formát je vhodný pro běžné použití, přestože není schopný zobrazit některé syté barvy.

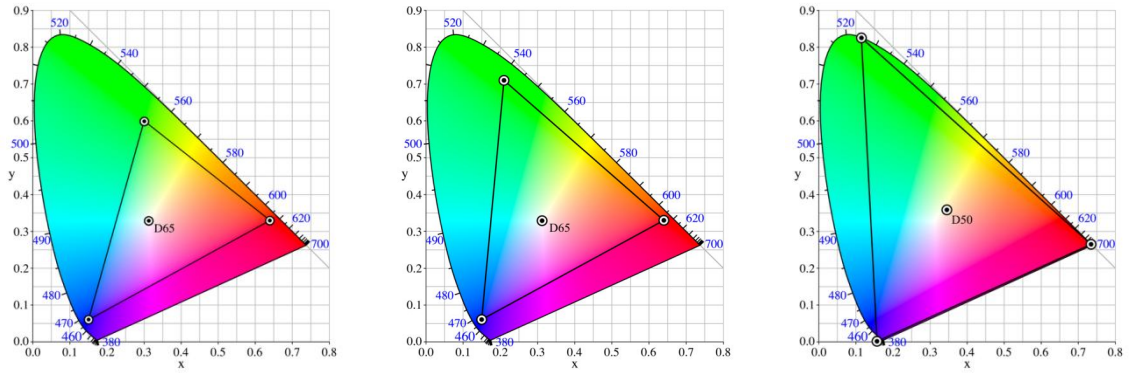
V aplikacích, kde je vyžadována lepší kvalita, je často používán *Adobe RGB* (obr. 2.9b), který má výrazně větší gamut[7]. V tomto barevném prostoru je možné zobrazit všechny tisknutelné barvy.

Velmi široký gamut má *Adobe Wide Gamut RGB* (obr. 2.9c) s vlnovými délkami základních barev 700 nm (R), 525 nm (G) a 450 nm (B). Dokáže zobrazit 77,6 % barev

[7] Zobrazitelná oblast barev v určitém barevném prostoru.

specifikovaných v barevném prostoru *CIELAB*, zatímco *Adobe RGB* pouze 50,6 %. Jeho nevýhodou je, že asi 8 % reprezentovatelných barev představují imaginární barvy, které není možné zobrazit na žádném fyzickém médiu a že zařízení pracující s tímto formátem je pravděpodobně nemožné vytvořit.

Bylo zavedeno mnoho dalších *RGB* standardů se zaměřením na fyzikální vlastnosti zařízení nebo se snahou o zobrazení co největší části viditelného světla. [9]



(a) sRGB (b) Adobe RGB (c) Adobe Wide Gamut RGB

Obr. 2.9: Porovnání velikosti gamutu barevných prostorů sRGB, Adobe RGB a Adobe Wide Gamut RGB. [12]

2.3.2. CMY, CMYK

Barevný systém *RGB* nelze použít tam, kde místo se světly pracujeme s tiskařskými barvami. Namísto něj se v takových aplikacích používá systém *CMY*, případně *CMYK*. Jedná se o systém subtraktivní, jehož základními složkami jsou azurová (C – cyan), purpurová (M – magenta) a žlutá (Y – yellow) barva.

Funguje na principu „odečítání“ barvy od bílého základu, na nějž se tiskne. Potřebujeme řídit množství odraženého červeného (R), zeleného (G) a modrého (B) světla. Na začátku máme bílý papír, který odráží veškeré světlo. Nanesením azurového (C) inkoustu měníme množství odraženého červeného světla, zelené a modré světlo se odráží beze změny. Podobně purpurový (M) inkoust mění množství odraženého zeleného světla a žlutý (Y) inkoust modrého světla.

Nanesením všech tří inkoustů by měl vzniknout povrch pohlcující všechny vlnové délky, jinými slovy černý. Vytvoření ideálního inkoustu, který by některé vlnové délky zcela pohlcoval a jiné zcela odrážel, je však pravděpodobně nemožné. Smícháním všech tří inkoustů tedy získáme barvu, která bude velmi tmavá, ale stále bude odrážet značnou část dopadajícího světla. Navíc bychom pro získání černé museli smíchat maximální

množství tří základních inkoustů, což by při tisku působilo značné technologické problémy, především při tisku na méně kvalitní papír.

Z toho důvodu byl ke třem barevným inkoustům přidán ještě čtvrtý, černý inkoust (K – blacK). Pro tisk černé plochy pak potřebujeme jediný inkoust, který navíc můžeme míchat s barevnými inkousty pro získání ještě tmavší barvy. [10]

2.3.3. CIE-XYZ

Výše zmiňované barevné systémy *RGB* a *CMYK* jsou relativní, jsou pevně svázané se zobrazovací technologií, pro kterou jsou určeny. Výsledek zobrazení je závislý na volbě primárních světel v *RGB* modelu, případně na volbě inkoustů v *CMYKu*. Toto nám nevadí, pokud příliš nezáleží na přesnosti zobrazovaných barev, například u televizního vysílání. Naopak v aplikacích, kde je přesnost zobrazení barvy kritická, jsou tyto barevné systémy nepoužitelné. Navíc potřebujeme systém, který dokáže vyjádřit všechny viditelné barvy za použití kladných složek, což nedokáže *RGB* ani *CMYK*.

Proto *Mezinárodní komise pro osvětlení CIE* zavedla nový barevný systém *CIE-XYZ*. Tento systém vychází z *RGB* modelu s vlnovými délkami primárních světel 700 nm (R), 546,1 nm (G) a 435,8 nm (B). Komise hledala takový transformovaný souřadný systém, aby se v něm dala zobrazit jakákoli barva kladnými koeficienty. Transformací, které tuto podmínku splňuje, existuje mnoho. *Komise CIE* zvolila pro převod vztahy (2.1).

$$\begin{aligned} X &= 0.49000*R + 0.31000*G + 0.20000*B \\ Y &= 0.17697*R + 0.81240*G + 0.01063*B \\ Z &= 0.00000*R + 0.01000*G + 0.99000*B \end{aligned} \tag{2.1}$$

Zpětný převod je definován vztahy (2.2)

$$\begin{aligned} R &= 2.36461*X - 0.89654*Y - 0.46807*Z \\ G &= -0.51517*X + 1.42641*Y + 0.08876*Z \\ B &= 0.00520*X - 0.01441*Y + 1.00920*Z \end{aligned} \tag{2.2}$$

Podrobnosti o převodu do barevného prostoru *CIE-XYZ* je možné najít v [11].

2.3.4. CIELAB

Systém *CIELAB* či zkráceně *L*a*b**, byl vytvořen jako systém napodobující způsob lidského vidění.

CIELAB je opět určen třemi hodnotami, a to světlostí (L – lightness) a dvěma kanály pro barevnou informaci. Kanál *a* určuje červeno-zelenou složku a kanál *b* žluto-modrou složku, tedy páry protikladných barev. I v lidském oku je barevná informace zpracovávána v obdobných barevných kanálech, proto neexistuje barva, kterou by člověk interpretoval jako červenozelenou či žlutomodrou. Smíšené barvy vznikají směsí těchto dvou kanálů.

CIELAB dokáže popsat všechny barvy viditelné lidským okem, ale i mnoho imaginárních barev, tedy barev, které není možné reprodukovat ve fyzickém světě. [12] Tento systém může být často výhodný pro barevné korekce obrázků, jelikož umožňuje upravovat jasový kanál, aniž by se projevily jakékoli změny v barvě, a naopak. [4]

2.3.5. HSV, HLS, HSI

Výše zmiňované systémy jsou výhodné pro různé aplikace, *RGB* pro zařízení vyzařující světlo, *CMYK* pro tisk, *CIE-XYZ* pro reprezentaci libovolné barvy bez svázanosti s konkrétním zařízením. Ani jeden z těchto systémů však není příliš uživatelsky příjemný. Člověk, který není v daném barevném prostoru příliš zběhlý, bude mít velké problémy s výběrem požadované barvy, či s výběrem barvy světlejší nebo sytější. Z toho důvodu vznikly barevné systémy orientované primárně na výběr barev, například *HSV* (Hue, Saturation, Value), *HLS* (Hue, Lightness, Saturation) nebo *HSI* (Hue, Saturation, Intensity).

Tyto systémy jsou také definovány trojicí hodnot, avšak místo s primárními barvami pracují s termíny odstín, sytost a světlost. Odstín se udává v rozsahu od 0° do 360°, určuje úhel na „barevném kruhu“. Jas se obvykle pohybuje v rozmezí od 0 do 1, představuje přechod od nejtmavších barev k nejsvětlejším. Sytost bývá udávána v procentech, kde 100% značí maximální sytost barvy. [10]

Algoritmy pro převod mezi systémy *HSV, HLS, HSI* a systémem *RGB* jsou popsány v [11].

2.3.6. Barevné atlasy

Jiným typem systémů pro výběr barev jsou tzv. *barevné atlasy* či *katalogy barev*, které se používají především v textilním, chemickém a tiskařském průmyslu. Jedná se

o tištěná pole barev, která jsou uspořádána podle odstínu, sytosti a jasu. Každému poli je přiřazen jednoznačný identifikátor, jímž je poté daná barva odkazována.

Mezi nejznámější patří *Munsellův katalog* nebo *Pantone.* [10]

2.3.7. Gama korekce

S problematikou barevných systémů silně souvisí gama korekce. Lidské oko nevnímá jas lineárně, v různých úrovních je schopné odlišit různé rozdíly jasu. Citlivější je na rozdíly v nižších úrovních jasu, změny jasu ve vyšších úrovních vnímá výrazně méně. Pokud bychom tedy diskrétní hodnoty obrázku převedli na jas monitoru lineárně, vnímali bychom přechod jasu mezi úrovněmi jako nerovnoměrný a v tmavých odstínech bychom viděli ostré skoky (tzv. posterizace) mezi jasy sousedních úrovní.

Existují dva způsoby kódování jasu – lineární (uniformní kvantizace) a nelineární (neuniformní kvantizace). Abychom se zbavili problému posterizace při použití lineárního kódování, potřebovali bychom pro reprezentaci jasu více než 8 bitovou hodnotu. Při neuniformním kódování jasu nepřevádíme hodnoty lineárně, ale pomocí mocninné funkce. Tím docílíme snížení rozdílů mezi tmavými úrovněmi a zvýšení rozdílů mezi světlými úrovněmi, přechody jasů výsledného obrázku budou působit plynule.

Při převodu relativního barevného systému (závislého na zobrazovacím zařízení) do absolutního je nutné před samotným převodem použít gama korekci (2.3), kde R je vstupní hodnota, R_k hodnota po aplikaci korekce a parametr γ je roven 2,5 (typická hodnota pro CRT monitory), abychom z výchozích nelineárních hodnot získali hodnoty lineární. Při opačném převodu musíme po převodu do relativního systému aplikovat gama korekci s hodnotou $\gamma = 1 / 2{,}5$, abychom získali nelineární hodnoty odpovídající lidskému vnímání jasu.

$$R_K = R^{\gamma} \tag{2.3}$$

Vztah (2.3) je pouze přibližným vzorcem, přesná definice gama korekce se liší pro různé barevné prostory, konkrétní vztahy jsou uvedeny například v [9].

2.4. Software pro barevné korekce

Při barevných korekcích je nutné, aby používaný software měl určité vlastnosti, které jsou potřebné pro efektivní a pohodlnou práci s ním. Jedná se především o

- Nástroj Křivky a další nástroje pro úpravy obrazu
- Nástroj Informace pro odečítání přesných hodnot pixelu před a po korekci
- Možnost sledovat během korekce náhled efektu na upravovaném obrázku (v reálném čase, nejlépe přes celou obrazovku)
- Možnost přepínání mezi náhledem korekce a stavem obrázku před korekcí
- Možnost měnit během korekce výřez náhledu nebo zvětšení obrázku
- Podporu ukládání úprav do historie (undo/redo)

Mezi další vlastnosti, které jsou u softwaru pro barevné korekce výhodné (ne však zcela nezbytné) patří

- Podpora různých barevných prostorů
- Podpora práce s vrstvami, režimu prolnutí vrstev
- Podpora efektových vrstev (vrstev obsahujících parametry určitého efektu, který se aplikuje na vrstvy pod ní)
- Podpora 16bitové grafiky
- Podpora selekcí
- Základní kreslicí a retušovací nástroje

Existuje mnoho programů, ve kterých je možné provádět více či méně pokročilé barevné korekce. V následujících kapitolách se zaměřím na hlavní výhody a nevýhody těch nejznámějších a nejpoužívanějších.

2.4.1. Adobe Photoshop

Adobe Photoshop je pravděpodobně nejznámějším profesionálním nástrojem pro úpravu obrázků, který poskytuje širokou škálu nástrojů pro barevné korekce.

Jeho počátky se datují do roku 1987, kdy student Michiganské univerzity Thomas Knoll začal psát program *Display* na zobrazování černobílých obrázků na monochromatickém monitoru. Spolu se svým bratrem Johnem program dále upravili a přejmenovali nejprve na *ImagePro* a později na *Photoshop*. V roce 1988 prodali licenci společnosti *Adobe* a o dva roky později spatřil světlo světa *Photoshop 1.0*, který byl určen pouze pro platformu Macintosh. První verzí fungující pod systémem

Windows byla verze 4.0 z roku 1996. V současné době je na trhu jedenáctá verze pod označením *CS4 (Creative Suite),* která byla vydána v roce 2008.

Photoshop podporuje práci v různých barevných prostorech, implementuje všechny základní nástroje pro barevnou korekci obrázků a mnoho dalších užitečných funkcí. Jeho největší nevýhodou je vysoká cena ($699[8] za *Photoshop CS4* nebo $999 za *Photoshop CS 4 Extended*) a velké hardwarové nároky.

2.4.2. Adobe Photoshop Elements

Adobe Photoshop Elements poskytuje podmnožinu funkcí standardního *Photoshopu*. Není určen pro profesionální barevné korekce, ale spíše pro amatérské korekce fotografií. Jeho cena je příznivější, *Adobe Photoshop Elements 7* se dá pořídit za $99. Nejvýraznějším nedostatkem tohoto softwaru je absence nástroje *Křivky*, který je zásadní pro kvalitní barevné korekce.

2.4.3. Adobe Photoshop Lightroom

Adobe Photoshop Lightroom je určen pro snadné spravování digitálních fotografií v počítači. Je určen především pro profesionální fotografy, kteří potřebují prohlížet a upravovat velké množství fotografií. Tento program poskytuje pouze omezený nástroj *Křivky* a jeho cena je vyšší než u *Adobe Photoshop Elements*, za tento produkt utratíte $299.

2.4.4. Corel Paint Shop Pro

Corel Paint Shop Pro je dalším z programů použitelných pro barevné korekce obrázků. Poskytuje kvalitní nástroje, ale oproti *Photoshopu* je o něco pomalejší a navíc existuje pouze ve verzi pro operační systém Windows. [12] *Corel Paint Shop Pro Photo X2 Ultimate* je prodáván za cenu $79,99.

2.4.5. Corel Photo-Paint

Corel Photo-Paint je rastrový grafický editor srovnatelný s *Photoshopem*, který je šířen v balíku *CorelDRAW Graphics Suite*. Na rozdíl od *Corel Paint Shop Pro* existuje i ve verzi pro Macintosh (nejnovější ve verzi 11) a Linux (ve verzi 9). Balík *CorelDRAW Graphics Suite X4* lze pořídit za $399.

[8] Všechny zmiňované ceny programů byly platné k datu 28. dubna 2009.

2.4.6. Gimp

Gimp (GNU Image Manipulation Program) je rastrový editor distribuovaný jako *open source*. Poskytuje velké množství kvalitních nástrojů pro barevné korekce i jiné úpravy obrázků. Bohužel však není možné ho použít pro předtiskovou přípravu obrázků, protože nepodporuje práci v barevném prostoru *CMYK*. Jeho další nevýhodou je, že neposkytuje paletu *Informace* pro odečítání přesných hodnot jednotlivých barevných kanálů, která je při barevných korekcích velice důležitá.

2.4.7. Irfanview

Irfanview je program šířený jako *freeware*[9], který je vhodný spíše k prohlížení fotografií, nástroje pro korekce poskytuje pouze zcela základní.

2.4.8. Další freewarové programy

Freewarových programů pro barevné korekce obrázků je možné najít velké množství, žádný z nich však neposkytuje najednou všechny potřebné vlastnosti zmíněné v úvodu kapitoly 2.4.

*Paint.NE*T podporuje práci s vrstvami, neomezené undo a širokou nabídku nástrojů, je ale určen pouze pro operační systém Windows. *Serif Photoplus 6* pracuje s vrstvami a nabízí řadu speciálních efektů. *PhotoFiltre 6* umožňuje pouze základní práci s obrázky bez podpory vrstev. *Pixia 4* pracuje s vrstvami a poskytuje velké množství filtrů pro úpravy obrázků. *XnView* je prohlížeč obrázků, který dokáže zobrazit obrázky 400 různých formátů a převést je do některého z 60 podporovaných formátů, navíc poskytuje základní nástroje pro úpravu obrázků. [15]

[9] Volně šiřitelný program, ke kterému se neposkytují zdrojové kódy a který je zakázáno vnitřně upravovat.

3. Realizační část

Předtím, než jsem začala vytvářet požadovanou aplikaci, bylo nutné zvolit vhodný programovací jazyk. Výsledná aplikace musí být přenositelná a musí být umožněno její šíření jako open source. Z charakteru aplikace dále plyne nutnost rychlé práce s jednotlivými pixely obrázku.

Z těchto požadavků vyplývá nemožnost použití některých programovacích jazyků. Jazyk C# nevyhovuje kvůli nepřenositelnosti výsledného programu, stejně jako Delphi. Java splňuje požadavky na přenositelnost, ale bohužel práce s jednotlivými pixely obrázku je neakceptovatelně pomalá, výsledná aplikace by byla v praxi téměř nepoužitelná. Z programovacích jazyků, které jsou v současnosti v praxi nejčastěji používány, zbývají jazyky C a C++. Oba jazyky umožňují vytvoření přenositelné aplikace a podporují přímý přístup k jednotlivým bytům obrázku, což bude zásadní při vytváření nástrojů pro barevné korekce. Pro implementaci jsem zvolila jazyk C++, který je objektově orientovaný a ke kterému existuje velká řada kvalitních knihoven.

Dále bylo nutné vhodně zvolit knihovnu pro vytváření grafického uživatelského rozhraní. Volila jsem mezi knihovnami FLTK (Fast, Light Toolkit), GTK+ (The GIMP Toolkit) a Qt. Knihovna FLTK je oproti zbylým dvěma výrazně menší a omezuje se čistě na funkčnost grafického rozhraní. GTK+ a Qt patří mezi nejpopulárnější knihovny pro vytváření GUI (graphical user interface), obě mají velké možnosti a splňují naše požadavky. Aplikaci budu vytvářet s využitím knihovny Qt, která je primárně určena pro jazyk C++ (na rozdíl od GTK+, která je napsána v jazyce C).

Knihovna Qt je pro nekomerční vývoj dostupná pod licencí LGPL[10], která umožňuje šíření výsledného software jako open source a poskytuje prostředky, které budu potřebovat při vytváření aplikace. Mezi nejznámější projekty využívající této knihovny patří KDE (K Desktop Environment), Opera, Google Earth, Skype, Adobe Photoshop Album,VirtualBox a OPIE (Open Palmtop Integrated Environment).

Informace o jazyku C++ a knihovně Qt, které jsem potřebovala při vytváření aplikace, jsem čerpala z [13] a [14].

Základní vztahy mezi třídami výsledné aplikace jsou symbolicky znázorněny na obr. A.10.

[10] Lesser General Public License je licence svobodného softwaru, která byla navržena jako kompromis mezi licencí GNU General Public License (GPL) a permisivními licencemi, jako jsou BSD licence nebo MIT License, více viz [12]

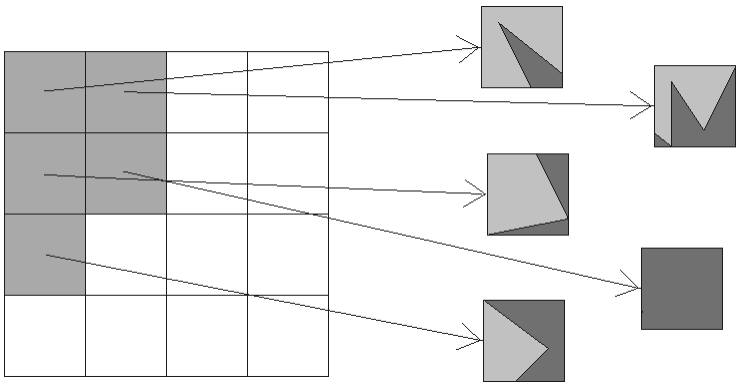
3.1. Datové struktury

3.1.1. Reprezentace obrázku

Nejprve bylo nutné navrhnout vhodné datové struktury pro uložení obrázku v paměti a práci s ním. Pro interaktivní práci s barevnými korekcemi je nezbytná rychlá odezva náhledu. V náhledu je ale zobrazena pouze část obrázku; např. při náhledu 1024x768 (0,75MPix[11]) z 15MPix obrázku potřebujeme pro náhled zpracovávat jen 5% obrazových dat. Při korekcích je také nutné mít možnost rychle srovnat obrázek před a po korekci. Pokud by byl obrázek uložen v paměti tak, jak byl načten, bylo by nutné při každé dílčí úpravě přepočítat hodnoty každého pixelu celého obrázku. Jinou možností by bylo přepočítávat pouze hodnoty, které potřebujeme (hodnoty pixelů ve vykreslovaném výřezu obrázku), ale pak by nebylo možné jednoduše zjistit, které pixely jsou platné a které musíme při posunu zobrazovaného výřezu překreslit, museli bychom tedy neustále přepočítávat hodnoty celé oblasti.

Proto obrázek reprezentuji ne jako jediný blok, ale jako dvourozměrné pole menších bloků pevné velikosti. Při provádění barevných korekcí stačí přepočítat hodnoty bloků, které spadají do aktuálně zobrazovaného výřezu. Stejně tak při přiblížení nebo oddálení není nutné transformovat celý obrázek, ale dopočtou se pouze hodnoty pixelů potřebných bloků. Díky tomu je překreslování plátna i zoomování mnohem rychlejší, než by bylo při práci s celým obrázkem.

Třídou reprezentující obrázek je *ImageMatrix*. Tato třída uchovává informaci o hodnotě zoomu, velikosti obrázku v daném přiblížení a o počtu bloků, ze kterých se obrázek skládá. Jejím nejdůležitějším atributem je dvourozměrné pole ukazatelů na jednotlivé bloky, které obsahují data obrázku, viz obr. 3.1.



Obr. 3.1: Reprezentace obrázku v paměti jako dvourozměrné pole ukazatelů na bloky

[11] Megapixel – 1 000 000 pixelů

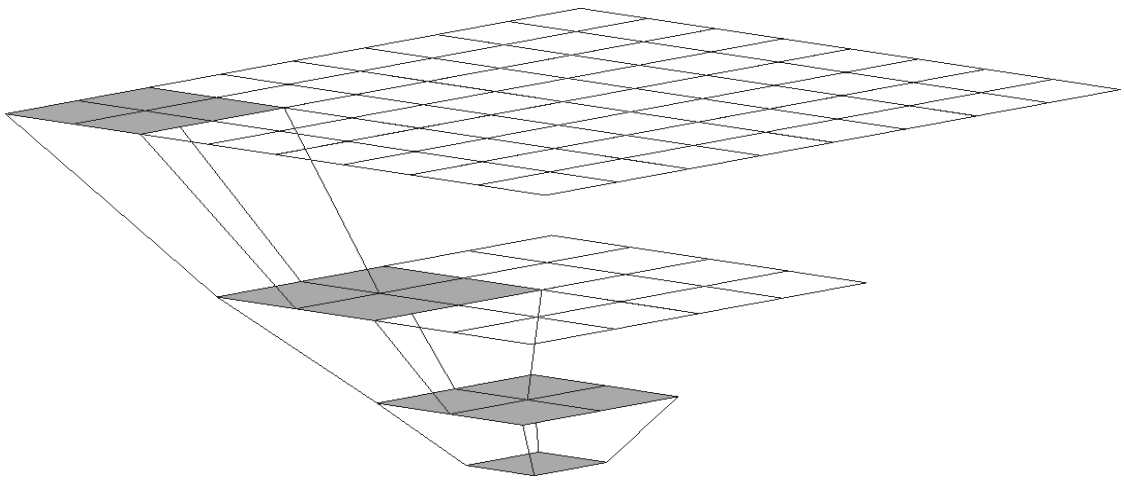
3.1.2. Blok obrázku

Jednotlivé bloky jsou instancemi třídy *ImageFraction*, která je odvozená od *QImage*, standardní třídy pro uložení obrázku. Od základní třídy se liší přidáním atributu *valid*, který určuje platnost bloku. Na základě tohoto atributu se při překreslování plátna rozhoduji, zda je nutné hodnoty bloku před vykreslením přepočítat.

Velikost bloku je výhodné volit jako mocninu 2. Při přibližování obrázku je pak možné jednoznačně určit, které pixely jsou potřebné pro vytvoření daného bloku. Více se této problematice budu věnovat v kapitole 3.1.3. Optimální velikostí bloku se ukázala být hodnota 128 x 128 pixelů, důvody, jež vedly k tomuto rozhodnutí, rozeberu podrobněji v kapitole 4.1.

3.1.3. Zoomovací pyramida

Při práci s obrázkem je často nutné jej opakovaně přiblížit či oddálit. Bylo by velice neefektivní, kdybychom při každé změně zoomu museli znovu transformovat obrázek. Proto v programu vytvářím zoomovací pyramidu. Jedná se o datovou strukturu, ve které je uložen jak obrázek v původním rozlišení, tak i jeho zvětšené či zmenšené verze, viz obr. 3.2.



Obr. 3.2: Zoomovací pyramida obrázku

Třída *ZoomPyramid* oddděděná od *QList* představuje spojový seznam, do kterého ukládám instance *ImageMatrix*. Při vytváření nové pyramidy předávám konstruktoru zdrojový obrázek, z kterého se vytvoří objekt *ImageMatrix* rozdělením obrázku na jednotlivé bloky. Tento objekt poté uložím do spojového seznamu a nastavím na jeho pozici iterátor *currentZoom*, který udržuje informaci o aktuální pozici v seznamu. Tento

iterátor využívám při vykreslování pro rozhodnutí, který obrázek z pyramidy mám použít. Na vytvořený objekt nastavím také ukazatel *imageOriginalSize*, který budu využívat při dopočítávání bloků v jiném přiblížení.

Požadujeme, aby zoomování probíhalo rychle a zároveň kvalitně, žádný algoritmus, který by zároveň splňoval oba požadavky, ale neexistuje, proto se musíme omezit pouze na určité stupně zoomu, které se provádějí jednodušeji. Zoomování proto probíhá pouze po násobcích 2, tzn. 200%, 400%, 800%, atd. pro přiblížení, případně 50%, 25%, 12,5% pro oddálení. To znamená, že při přiblížení se jeden pixel zobrazí na oblast 2x2 pixely a při oddálení z oblasti 2x2 pixely vznikne jeden pixel zmenšeného obrázku. Z toho vyplývá, že z jednoho bloku vzniknou čtyři bloky v úrovni s dvojnásobným zoomem a že ze čtyř bloků vznikne jediný blok v úrovni s poloviční zoomem, viz obr. 3.2.

Při přiblížení obrázku nejprve zkontroluji, zda v seznamu existuje další prvek, pokud ano, pouze na něj přesměruji iterátor *currentZoom*. Pokud další prvek v seznamu neexistuje, musím jej vytvořit. Vytvořím nový objekt *ImageMatrix*, jehož matice pro bloky obrázku bude mít dvojnásobné rozměry než má matice v aktuálním přiblížení. Podobně při oddálení obrázku kontroluji, zda existuje předcházející prvek a pokud ano, přesměruji na něj iterátor. V opačném případě vytvořím objekt *ImageMatrix* s maticí rozměrů $(n / 2 + 1)$ x $(m / 2 + 1)$, kde n je počet sloupců matice v aktuálním přiblížení a m počet jejích řádků – přičtení jedničky zaručí, že matice bude mít dostatečnou velikost pro uložení obrázku i v případě, že n nebo m budou lichá čísla. Ukazatele na bloky budou prozatím nastaveny na NULL, jejich hodnoty dopočítám pouze pokud budou potřebné při vykreslování.

Nové bloky dopočítávám z obrázku v původním rozlišení, odpovídající bloky v přilehlé úrovni pyramidy nemusí být totiž vždy k dispozici. Například při přiblížení obrázku na 800% se ve všech úrovních dopočtou bloky potřebné pro vykreslení aktuálního výřezu. Pokud se však uživatel nyní v obrázku posune, musím dopočítat potřebné bloky, jejichž ekvivalenty v nižší úrovni nemusí být dostupné. Bylo by zbytečné je v takové situaci dopočítávat, když můžeme snadno využít data z obrázku se zoomem 100%, která jsou k dispozici vždy.

Ve chvíli, kdy při vykreslováním narazím na blok, který vůbec neexistuje nebo je neplatný, jej musím dopočítat. Pokud se jedná o blok v zoomu z větším než 100%, potřebuji jeden blok obrázku v zoomu 100% na dopočtení $(z / 100)^2$ bloků obrázku v zoomu z. Například při zoomu 800% tedy z jednoho bloku základního obrázku získám 8x8 bloků zvětšeného obrázku. Obzvlášť při větším přiblížení by bylo zbytečné

dopočítávat všechny příslušné bloky, z nichž velkou část vůbec nepotřebujeme. Lepším řešením je vypočítat polohu potřebných pixelů v základním bloku a vytvořit vždy pouze jeden konkrétní nový blok, který je nutný při vykreslování. Právě kvůli tomuto přístupu k zoomování je výhodné volit velikost bloku jako mocninu 2, při jiné velikosti by nebylo možné přesně určit odpovídající pixely potřebné pro vytvoření nového bloku.

Při vytváření bloku v zoomu z menším než 100% je postup obdobný. Pro vytvoření jednoho bloku nyní potřebuji odpovídajících *(100 / z) x (100 / z)* bloků základního obrázku. Například při zoomu 25% tedy potřebuji 4x4 bloky základního obrázku, z nichž získám jeden požadovaný blok. Tuto metodu by bylo vhodné rozšířit tak, aby využívala bloky přilehlé úrovně, jsou-li k dispozici.

Pro samotné zvětšení či zmenšení bloku obrázku používám metodu knihovny Qt *QImage scaledToHeight(int h, enum TransformationMode mode)*, které předám požadovanou výšku výsledného obrázku. Toto řešení je pouze dočasné, metoda optimalizovaná pro potřeby naší aplikace (počítání pouze dvojnásobného nebo poloviční rozlišení) by měla poskytovat lepší výsledky.

3.2. Aplikační logika

3.2.1. Implementace různých barevných prostorů

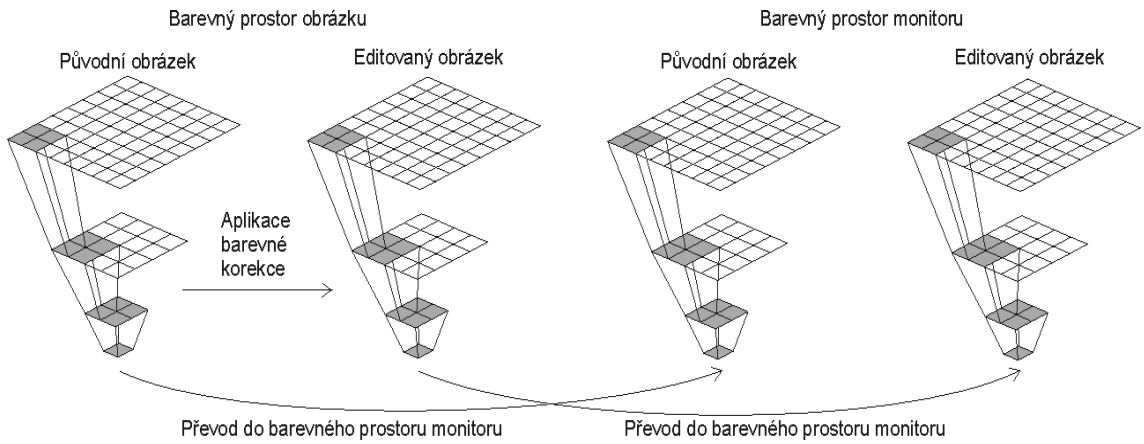
Architektura aplikace je přizpůsobená pro budoucí podporu práce s obrázky v různých barevných prostorech. Abstraktní třída *AbstractImage* poskytuje funkčnost, která není závislá na konkrétním barevném prostoru. Jedná se především o metody *zoomIn()* a *zoomOut()* pro přiblížení nebo oddálení obrázku a metodu *ImageMatrix *getImageMatrixZoomed()*, která vrací ukazatel na obrázek v aktuálním zoomu potřebný při vykreslování. Dále definuje signaturu čistě virtuálních metod pro barevné korekce, které budou implementovány v potomcích *AbstractImage*. Tyto metody nemohou být společné pro obrázky v různých barevných prostorech, protože budou manipulovat s jednotlivými pixely obrázku. Způsob uložení obrázku v paměti se může u různých prostorů lišit a z toho vyplývá i nutnost použití jiné implementace metody pro barevnou korekci.

Třídy odvozené od *AbstractImage* jsou určeny pro práci s obrázkem v konkrétním prostoru, např. *RGBImage* pro *RGB* obrázky, *CMYKImage* pro *CMYK* obrázky, *LABImage* pro *CIELAB*, atd. Aplikace v současné době poskytuje pouze třídu *RGBImage*, implementace podpory ostatních barevných systémů by byla obdobná.

Způsob práce jednotlivých metod třídy *RGBImage* blíže popíši v kapitole 3.3, při popisu implementace nástroje pro danou barevnou korekci.

Načtený obrázek tedy obecně nemusí být v barevném prostoru *RGB*, ale například v *CMYKu* nebo *CIELABu*. Obrázek v takovém barevném prostoru není možné přímo zobrazit na monitoru, je nutné jej nejprve převést do *RGB* prostoru monitoru. I *RGB* obrázek je někdy nutné před zobrazením převést, tato situace nastává v případě, že obrázek využívá jiného standardu než monitor.

Z tohoto důvodu je nutné, aby každá instance třídy *AbstractImage* vlastnila čtyři objekty *ZoomPyramid,* dvě pro obrázek v originálním barevném prostoru a dvě pro obrázek v prostoru monitoru. Jedna z pyramid vždy obsahuje data původního obrázku před aplikací barevné korekce a druhá je určena pro editovaný obrázek. Postup při korekci je pak následující: v pyramidě původního obrázku v originálním barevném prostoru se vybere právě zpracovávaný blok, na který se použije funkce pro danou barevnou korekci. Vzniklý blok se uloží do pyramidy pro editovaný obrázek. S využitím metod třídy *ColorConvertor* se blok transformuje do barevného prostoru monitoru a výsledný blok se uloží do pyramidy pro editovaný obrázek v prostoru monitoru (obr. 3.3). Kterou z pyramid monitoru použít při vykreslování obrázku se rozhoduji na základě toho, zda je zapnutý či vypnutý náhled v právě používaném nástroji pro barevné korekce. Pokud je náhled zapnutý, vykresluji editovaný obrázek, pokud je vypnutý nebo pokud není otevřený žádný nástroj pro korekce, vykresluji původní obrázek. Díky tomu, že mám pyramidu pro původní i editovaný obrázek, je možné mezi oběma verzemi velice rychle přepínat.



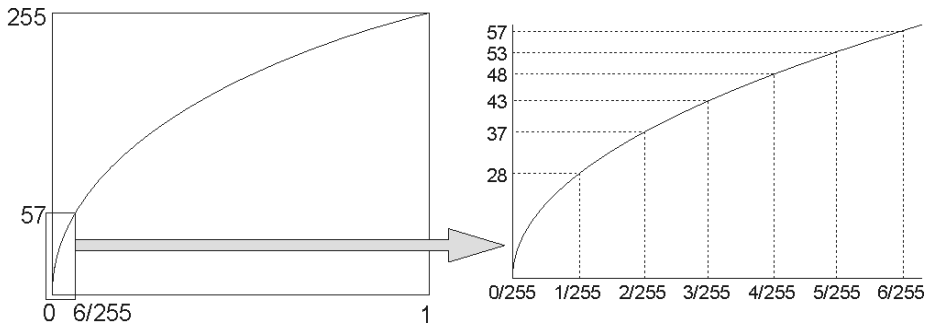
Obr. 3.3: Zoomovací pyramidy obrázku a vztahy mezi nimi

3.2.2. Převody mezi barevnými prostory

Pro převody mezi barevnými prostory slouží třída *ColorConvertor*. Její metody *setFrom(enum Colorspace)* a *setTo(enum Colorspace)* slouží k nastavení výchozího a cílového barevného prostoru. Po patřičné inicializaci těchto atributů je možné použít metodu *convert()* pro převod mezi zadanými prostory. Metodě se předají hodnoty převáděného pixelu v jednotlivých kanálech a ona na základě nastavených atributů rozhodne, kterou převodní funkci použít. Vytvářet převodní funkce pro převod mezi každými dvěma podporovanými barevnými prostory by bylo zbytečné, využívám zde převodu z výchozího barevného prostoru do prostoru *CIEXYZ* a z *CIEXYZ* do cílového prostoru.

Při převodech *RGB* systémů je nutné počítat s gama korekcí. Při převodu do *CIEXYZ* na vstupní hodnoty nejprve aplikuji gama korekci a poté přenásobením odpovídající převodní maticí získám požadované hodnoty. Při opačném převodu nejprve *CIEXYZ* hodnoty přenásobím maticí pro převod do cílového prostoru a teprve pak aplikuji gama korekci. Hodnoty převodních matic jsem čerpala z [9].

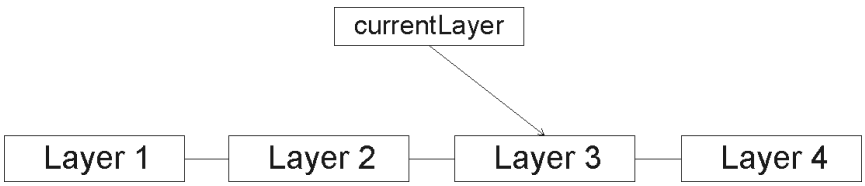
Vztahy pro gama korekci obsahují operaci umocňování, která je velice časově náročná, při počítání gama korekce podle vztahu $R_K = R^{\gamma}$ trval převod oblasti 400x300 pixelů řádově vteřiny. Proto v konstruktoru objektu *ColorConvertor* vytvářím vyhledávací tabulky pro gama korekci a při výpočtu pouze použiji odpovídající předpočítanou hodnotu, s tímto postupem převod stejně velké oblasti zabere řádově desítky milisekund. Při převodu do *CIEXYZ* převádím diskrétní *RGB* hodnotu z množiny {0, …, 255} na *CIEXYZ* hodnotu z intervalu <0, 1>, stačí mi tedy pole 256 hodnot – *RGB* hodnotu použiji jako index v poli a prvek pole bude odpovídající *CIEXYZ* hodnota. Situace při zpětném převodu je obtížnější – hodnoty intervalu <0, 1> potřebuji převést na diskrétní hodnoty z {0, …, 255}, nelze je tedy využít přímo jako index do pole. Převod navíc není lineární, proto nemůžeme jednoduše vynásobit vstupní hodnotu číslem 255 a výsledek použít jako index do pole, tento přístup by vedl k obrovským nepřesnostem především pro vstupní hodnoty blížící se nule (viz obr. 3.4). Tento problém jsem vyřešila použitím tří polí různé velikosti a tedy i různé přesnosti pro jednotlivé intervaly. Pro vstupní hodnoty do 0,0001 používám pole s největší přesností, pro hodnoty do 0,02 druhé pole s o něco menší přesnosti a pro ostatní hodnoty pole s nejmenší přesností. Tento přístup vede k dobrým výsledkům.



Obr. 3.4: Demonstrace problému vytváření vyhledávací tabulky pro gama korekci

3.2.3. Vrstvy

Aplikace umožňuje vytváření obrázku z více vrstev. Každá instance *AbstractImage* pak představuje jednu vrstvu kompozitu, relace mezi jednotlivými vrstvami určuje třída *Composite*. Třída *Composite* je odvozená od *QList*, ukládá tedy jednotlivé vrstvy do spojového seznamu. Vrstva na začátku seznamu je základní vrstvou, další vrstvy následují v pořadí, v jakém se nacházejí v kompozitu, viz obr. 3.5. K určení pozice aktuálně upravované vrstvy slouží iterátor *currentLayer*.



Obr. 3.5: Kompozit skládající se ze čtyř vrstev. „Layer 1“ je základní vrstvou, úpravy se provádí na vrstvě „Layer 3“, která je druhou vrstvou odshora.

Do kompozitu je možné vložit novou vrstvu dvěma způsoby. Prvním je vytvoření kopie zvolené vrstvy metodou *AbstractImage *duplicateLayer()*, tato metoda vytvoří novou instanci konkrétního potomka *AbstractImage* a zkopíruje do ní data původního obrázku. Druhým způsobem je použití metody *AbstractImage *newLayer()*, která vytvoří novou vrstvu vyplněnou barvou, kterou si uživatel zvolí v příslušném dialogu. Vytvořená vrstva bude mít vždy stejnou velikost jako základní vrstva. K vymazání zvolené vrstvy slouží metoda *removeLayer()*.

Každá vrstva má atribut *id*, který ji jednoznačně identifikuje – tento atribut se přiřazuje vrstvě při jejím vytvoření a je potřebný pro správnou funkčnost historie úprav, viz kapitola 3.2.6.

3.2.4. Vykreslování kompozitu

Vrstvy kompozitu mohou být částečně průhledné, vzhled kompozitu vrstev tedy ovlivní všechny průhledné vrstvy a první neprůhledná, počítáno od nejvyšší vrstvy. Vrstvy nacházející se pod neprůhlednou vrstvou nemají šanci vzhled výsledného obrázku ovlivnit, bylo by proto zbytečné je vykreslovat.

Vykreslování probíhá v metodě *paintComposite(QPainter &painter, QRect rect, int widthVisibleArea, int heightVisibleArea)* třídy *Composite*, tato metoda je volána při každém překreslování plátna *Canvasu*. *Painter* je reference na objekt, který umožňuje kreslení na *Canvas*, zbylé tři parametry slouží k identifikaci oblasti, kterou je nutné překreslit.

Před samotným vykreslováním najdu v kompozitu nejvyšší viditelnou neprůhlednou vrstvu a od ní začnu vykreslovat. Vykreslení každé vrstvy probíhá v těchto krocích:

- Zkontroluji, zda je vrstva viditelná a pokud není, pokračuji s vykreslováním následující vrstvy.
- Získám matici s bloky obrázku v odpovídajícím přiblížení voláním metody *ImageMatrix *getImageMatrixZoomed()*. Tato metoda se zároveň postará o aplikaci správné barevné korekce, pokud právě probíhá úprava obrázku, více v kapitole 3.2.5.
- Z parametrů identifikujících vykreslovanou oblast zjistím počet bloků obrázku, které budu potřebovat, a souřadnice levého horního rohu překreslované oblasti.
- Postupně procházím jednotlivé bloky a vykresluji je na odpovídající místo *Canvasu*. O správné zobrazení průhlednosti se stará sám *QPainter* díky nastavení atributu *setCompositionMode(QPainter::CompositionMode_SourceOver)*. Používání této metody je pouze dočasné, pro další urychlení vykreslování bude nutné implementovat vlastní metodu, která se postará o správné smíchání barev v paměti a vykreslení výsledku.
- Po vykreslení všech bloků pokračuji další vrstvou.

3.2.5. Barevné korekce

Všechny barevné korekce fungují na společném principu. Ovlivňováním parametrů příslušného nástroje se mění hodnoty pixelů v jednotlivých kanálech podle vzniklé převodní funkce. Pixel v každém kanálu může nabývat pouze diskrétních hodnot

z intervalu 0 až 255[12], jakoukoli barevnou korekci tedy můžeme definovat polem 256 hodnot pro každý barevný kanál a jedním pro kompozitní kanál; vstupní hodnota je vždy indexem v daném poli. V případě současné úpravy kompozitního kanálu i jednotlivých barevných kanálů je nutné nejprve použít výchozí hodnotu pixelu v daném barevném kanálu jako index do pole pro kompozitní kanál a získanou hodnotu následně jako index do pole pro odpovídající barevný kanál.

Trochu jiná situace je u barevných korekcí, které nezávisí jen na hodnotě konkrétního barevného kanálu, ale i na hodnotě ostatních kanálů daného pixelu, jako například u nástroje *Práh*, kde výsledné hodnoty závisí na jasu pixelu. V takovém případě musíme pro každý pixel počítat hodnotu jasu, další postup je však shodný, zjištěnou hodnotu opět použijeme jako index do převodního pole.

Pokud tedy budou všechny nástroje pracovat na tomto principu a správně nastavovat hodnoty převodních polí, bude možné pro všechny korekce pracující s kanály odděleně použít jedinou metodu pro manipulaci s pixely a pro korekce pracující s jasem metodu druhou. Rozhodnutí, kterou z metod použít, probíhá na základě nastavení atributu *ModifyingAction action*, kde *ModifyingAction* je výčet možných typů barevných korekcí. Při potřebě implementovat nástroj pro barevnou korekci, pro kterou nelze použít ani jeden z výše zmiňovaných způsobů, lze snadno doplnit další typ barevné korekce do výčtu a přiřadit jí metodu, která bude splňovat požadavky dané korekce.

Při požadavku na překreslení oblasti se rozhodne, zda je nutné aplikovat korekci, podle nastavení atributů *preview* a *modified*. *Preview* je nastaveno na *true* pokud je otevřený některý z nástrojů a zapnutý náhled úprav. *Modified* se nastaví na *true* po provedení úpravy obrázku, tento atribut zabrání zbytečnému přepočítávání zatím neupraveného obrázku. Korekce se aplikuje pouze pokud oba atributy mají hodnotu *true*.

Způsob manipulace s pixely ukážu na příkladu třídy *RGBImage* a metodě *applyCurves()*. Tato metoda slouží pro aplikaci křivek nástroje *Curves*, ale díky tomu, že pracuje pouze s hodnotami převodních polí, je možné stejnou metodu použít pro jakoukoli korekci, která pracuje s barevnými kanály odděleně. Metoda je přetížená pro převod celého obrázku nebo pouze pro převod viditelné oblasti, implementace se liší pouze způsobem určení, které bloky je nutné přepočítat. Metoda pro převod viditelné oblasti se využívá při provádění korekce se zapnutým náhledem, čímž se zabrání

[12] Platí pro 32 bitové barvy, kde na každý kanál připadá 8 bitů.

zbytečnému přepočítávání oblastí, které jsou pro nás v tu dobu nepodstatné. Druhá metoda se používá po potvrzení korekce pro přepočítání celého obrázku. Metody prochází všechny určené bloky a na každém provedou kód (3.1).

```
// pokud je blok v barevném prostoru monitoru platný, pokračuj s další iterací
if (screen->isValid(i, j)) continue;

// získaní bloku na aktuálním indexu
QImage fr = temp->getImageFraction(i, j);

int counter = 0;      // nastavení počítadla bytů na nulu

quint8 *p = (quint8*)fr.bits();      // získání ukazatele na první byte bloku
quint8 *end = p + fr.numBytes();     // získání ukazatele na poslední byte bloku

while ( p < end )     // průchod celého obrázku po bytech
{
       if (counter % 4 == 2) {                    // červená složka
              // Získaní výsledné hodnoty barevného kanálu z převodních polí.
              // transitFunction[0] je pole pro převod kompozitu všech kanálu,
              // hodnota získaná z tohoto pole pak slouží jako index
              // do převodního pole konkrétního kanálu
              *p = transitFunctionCurves[1][transitFunctionCurves[0][*p]];
       } else if (counter % 4 == 1) {       // modrá složka
              *p = transitFunctionCurves[2][transitFunctionCurves[0][*p]];
       } else if (counter % 4 == 0) {       // zelená složka
              *p = transitFunctionCurves[3][transitFunctionCurves[0][*p]];
       }
       p++;           // posun na další byte
       counter++;     // zvýšení počítadla bytů

}
```

(3.1): Ukázka kódu pro manipulaci s pixely při barevných korekcích.

3.2.6. Historie

K ukládání posloupnosti úprav obrázku slouží třída *History*, která je odvozená od *QListu*. Vždy po provedení úpravy nebo vytvoření nové vrstvy uložím na konec spojového seznamu kopii příslušné instance *AbstractImage*. Nelze provádět úpravy na více vrstvách současně, proto je postačující uložit pouze nový stav upravované vrstvy, zbytek kompozitu zůstává nezměněn.

Při uložení vrstvy z ní vymažu nepotřebná data, která by zbytečně zabírala paměť – pyramidy pro editovaný obrázek a všechny úrovně pyramid s původním obrázkem vyjma úrovně s obrázkem ve skutečné velikosti.

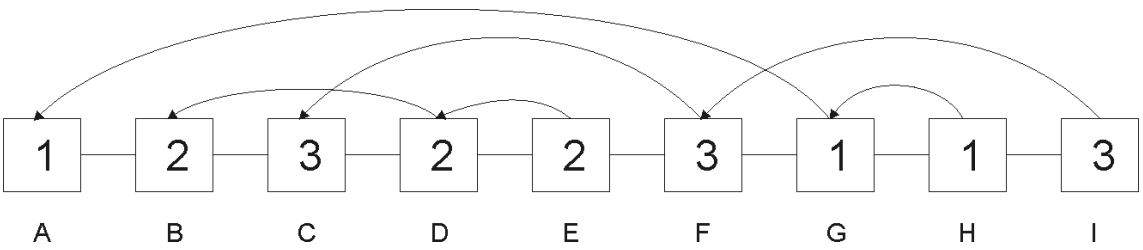
History obsahuje iterátor ukazující na současnou pozici, při procházení historie jej posunuji podle směru průchodu. Při návratu v historii potřebuji obnovit data z předchozího stavu vrstvy v kompozitu, o kterou vrstvu kompozitu se jedná určím podle *id* vrstvy. Pokud vrstva nemá předchozí stav, znamená to, že se nacházíme v bodě, kdy byla vrstva vytvořena, a při návratu v historii ji musíme z kompozitu

vymazat. Pokud vrstva má předchozí stav, ale nenachází se v kompozitu, jsme v bodě, kdy byla odpovídající vrstva vymazána a při návratu ji musíme v kompozitu obnovit.

Při posunu v historii dopředu je situace o něco jednodušší. Úpravy jsou v historii zaznamenány v pořadí, jak byly prováděny, proto stačí vždy obnovit data vrstvy podle instance *AbstractImage*, která se nachází ve spojovém seznamu na další pozici. Pokud narazíme na vrstvu, která se v kompozitu nenachází, jsme v bodě vytvoření nové vrstvy a při posunu dopředu je nutné ji obnovit. K označení vymazání vrstvy do historie ukládám instanci *AbstractImage*, která neobsahuje žádná obrázková data, ale pouze svoje *id* pro určení, která vrstva má být smazána. Při nalezení takovéto vrstvy v historii z kompozitu vymažu vrstvu s odpovídajícím *id*.

Na obrázku 3.6 je znázorněna historie úprav obrázku se třemi vrstvami, číslo v rámečku představuje *id* vrstvy. Z obrázku je možné vyčíst, že úpravy probíhaly takto: nejprve byl otevřen nový obrázek, kterému bylo přiřazeno *id* 1. Poté byly vytvořeny nové vrstvy s *id* 2 a 3. Následně byly provedeny dvě úpravy vrstvy 2, jedna úprava vrstvy 3, dvě úpravy vrstvy 1 a nakonec jedna úprava vrstvy 3.

Nacházíme-li se ve stavu I a chceme-li vrátit zpět poslední úpravu, je nutné ve vrstvě 3 v kompozitu obnovit data, která jsme do historie uložili ve stavu F. Po provedení této akce se budeme nacházet ve stavu H – při dalším návratu v historii je tedy nutné ve vrstvě 1 obnovit data uložená do historie ve stavu G.



Obr. 3.6: Historie. Úpravy obrázku se ukládají do spojového seznamu, každá úprava ukazuje na svůj předchozí stav, pokud v seznamu existuje.

3.3. Grafické uživatelské rozhraní

Grafické uživatelské rozhraní je vytvořeno výhradně s využitím komponent knihovny Qt. Hlavní okno programu tvoří třída *MainWindow*. Toto okno obsahuje hlavní menu programu, toolbary s nejpoužívanějšími funkcemi a oblast pro komponentu *QMdiArea*. Tato komponenta obsahuje oblast pro otevření více dokumentů stejného

typu a poskytuje základní funkce potřebné pro jejich správu, např. se stará o přepínání mezi okny, o rozložení oken v dané oblasti, zavření aktivního okna, atd.

3.3.1. Canvas

Podokno komponenty *QMdiArea* je tvořeno objektem *MyScrollArea*, který představuje oblast s vertikálním a horizontálním posuvníkem. Od standardní *QScrollArea* se liší pouze tím, že má překrytou metodu *closeEvent()*, která je volána při zavírání okna. V této metodě kontroluji, zda byly v obrázku provedeny změny, a pokud ano, vyzvu uživatele k jejich uložení.

Centrálním widgetem oblasti s posuvníky je *Canvas* – plátno, na které je vykreslován upravovaný obrázek.

3.3.2. Informace

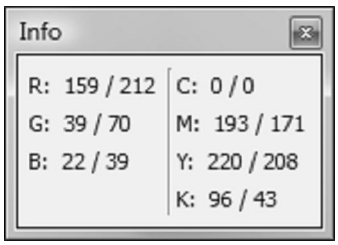
Paleta Informace je implementována třídou *Palette*. Stejně jako u všech ostatních nástrojů v konstruktoru nastavuji atribut *setWindowFlags(Qt::Tool)*, toto nastavení zaručí, že se okno bude chovat tak, jak od něj očekáváme, tedy například při minimalizaci či obnovení hlavního okna budou na událost reagovat i otevřené nástroje.

Tento nástroj umožňuje přesné odečítání číselných hodnot barev pixelu pod kurzorem myši. Někdy je výhodnější nezjišťovat přesnou hodnotu konkrétního pixelu, ale spíše průměrnou hodnotu v určitém okolí, k tomuto nastavení slouží položka *Sample Size* v hlavním menu programu.

Při pohybu myši nad *Canvasem* je z metody *mouseMoveEvent()* volána *setPixelInfo(int x, int y)*, která se postará o získání průměrných hodnot v požadovaném okolí kurzoru. Samotný výpočet probíhá ve virtuální funkci *int *getPixelValues(int x, int y, int sampleSize, bool edit)* třídy *AbstractImage*. Podle předávaného parametru *edit* se určuje, zda se mají hodnoty získat z původního či editovaného obrázku, *x* a *y* určují pozici požadovaného pixelu. Při výpočtu v cyklu procházím příslušné pixely a jejich hodnoty přičítám do pomocných proměnných, výslednou hodnotu získám vydělením pomocných hodnot počtem zpracovaných pixelů. Návratovou hodnotou je pole výsledných hodnot v jednotlivých barevných kanálech.

Pokud je otevřený některý z nástrojů pro barevnou korekci a je zapnutý náhled, bude se tato funkce volat dvakrát, poprvé pro zjištění původních hodnot pixelu (parametr *edit* bude false) a následně pro zjištění hodnot po příslušné úpravě (*edit* bude

true). Tyto dvě hodnoty se budou zobrazovat v levé polovině nástroje, oddělené lomítkem (viz obr. 3.7).

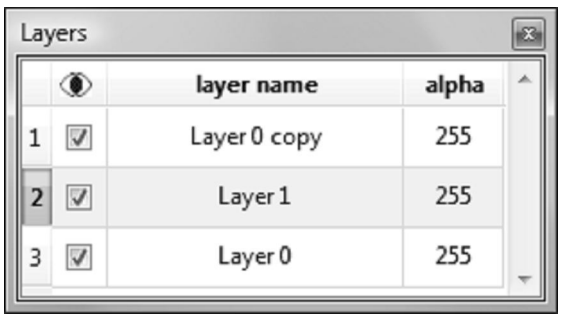


Obr. 3.7: Nástroj Informace

V pravé polovině nástroje je možné zobrazit ekvivalentní hodnoty v jiném barevném prostoru, v současnosti nástroj umožňuje výběr mezi *RGB, CMYK* a *HSV*. Převod do požadovaného systému probíhá v třídě *Palette* v metodě *setSecondaryChannelInfo(int *orig, int *edit)* s využitím standardních metod pro převod, které poskytuje knihovna Qt. Po doplnění potřebných převodních funkcí do třídy *ColorConvertor* bude možné je podobným způsobem využít pro převody do dalších barevných prostorů, které Qt standardně nepodporuje.

3.3.3. Vrstvy

Třída *Layers* poskytuje grafické uživatelské rozhraní pro práci s vrstvami. Třída je odvozená od komponenty *QTableWidget*, jednotlivé vrstvy jsou tedy zobrazovány v tabulkové struktuře (obr. 3.8).



Obr. 3.8: Nástroj pro manipulaci s vrstvami.

Nástroj pracuje s objektem *Composite*, který přísluší upravovanému obrázku. Odkaz na tento objekt je nástroji předán při volání metody *updateLayers(Composite *composite)*, která je volána při každém přepnutí na jiný obrázek a která se stará o správnou aktualizaci obsahu nástroje.

Při stisknutí pravého tlačítka myši nad nástrojem je v metodě reagující na tuto událost vytvořeno kontextové menu, které obsahuje položky pro přidání nové vrstvy, zkopírování vybrané vrstvy nebo její smazání. Po vybrání jedné z možností je zavolána příslušná metoda třídy *Composite*, která provede požadovanou akci, a následně je aktualizován obsah nástroje, aby odpovídal současnému stavu.

Nástroj umožňuje změnu pořadí vrstev přetažením myší na jinou pozici, tato funkčnost je vytvořena s využitím mechanismu drag and drop. Při stisku levého tlačítka myši si zaznamenám pozici kurzoru, při dalším pohybu myši pak počítám vzdálenost kurzoru od zaznamenané polohy. Jakmile vzdálenost překročí mez pro zahájení dragu[13], zavolá se metoda *startDrag()*, která vytvoří nový objekt *QDrag*, tento objekt obsahuje informace o tažené vrstvě. Po uvolnění tlačítka myši se vyvolá metoda *dropEvent(QDropEvent *event)*, v níž se zkontroluje platnost nové pozice a pokud je platná, vrstva se na ni přesune. Pro správnou funkčnost je nutné, aby nástroj přijímal pouze objekty přetahovaných vrstev, k tomuto nastavení slouží metody *dragEnterEvent(QDragEnterEvent *event)* a *dragMoveEvent(QDragMoveEvent *event)*

3.3.4. Násobení kanálu

Grafické uživatelské rozhraní nástroje pro násobení kanálu poskytuje třída *Sliders*. Každý barevný kanál má přiřazený posuvník, změnou jeho hodnoty měníme konstantu, kterou bude vynásoben příslušný barevný kanál obrázku. K uložení těchto konstant slouží pole *sliderFactors* ve třídě *AbstractImage*. Při každé změně posuvníku se vyvolá signál propojený se slotem třídy *MainWindow*, který se postará o správné nastavení hodnoty *sliderFactors*.

Po změně hodnoty pole je nutné nastavit atribut *ModifyingAction action* na *SLIDERS*, což zaručí aplikaci správné barevné korekce při překreslování obrázku. Manipulaci s pixely má na starosti metoda *multiplyChannel()*, která vynásobí každou barevnou složku příslušnou konstantou. Tato metoda slouží spíše jako ukázka jednoho možného přístupu k manipulaci s pixely, v praxi by bylo vhodnější využít přístupu vysvětleného v kapitole 3.2.5, tedy vytvoření vyhledávací tabulky s převodními hodnotami.

[13] Tato hodnota se může lišit u různých platforem, k jejímu zjištění slouží metoda knihovny Qt *QApplication::startDragDistance()*. Zahájení dragu až po překročení této meze zabrání tažení při neúmyslném stisknutí myši.

K potvrzení úpravy obrázku slouží tlačítko *Ok*, po jeho stisknutí se úprava uloží do historie a nástroj se zavře.

3.3.5. Křivky

Nástroj křivky tvoří dvě třídy, *Curves* a *CurvesCanvas*. *Curves* je hlavním oknem nástroje, obsahuje ovládací prvky pro zvolení barevného kanálu, zapnutí či vypnutí náhledu, přepínání způsobu interpolace křivky a potvrzení úpravy či její zrušení. *CurvesCanvas* pak tvoří plátno, na které je vykreslován histogram upravovaného obrázku a křivka charakterizující převodní funkci (viz kapitola 2.2.4). Nástroj pracuje s vrstvou, která byla zvolena při otevření nástroje.

Po otevření nástroje nebo po přepnutí na jiný barevný kanál se v metodě *recalculateHistogram()* vypočítají hodnoty pro příslušný histogram, tyto hodnoty se ukládají do pole *histogram*. Postup výpočtu je přímočarý – procházím obrázek a pro každý pixel zjistím hodnotu ve zvoleném kanálu, případně vypočtu hodnotu jasu pro kompozitní kanál, a poté inkrementuji hodnotu pole *histogram* na indexu odpovídajícím získané hodnotě. Nakonec celé pole projdu a zjistím maximální hodnotu, kterou využiji pro správné upravení proporcí histogramu.

Metodou *createBackgroundImage()* vykreslím histogram do obrázku. Tento obrázek budu vykreslovat na pozadí plátna a tím se vyhnu opakovanému vykreslování celého histogramu při každém překreslování nástroje.

Vložení nového bodu do křivky probíhá v metodě *mousePressEvent()*, která reaguje na stisk tlačítka myši nad oblastí plátna. Nejprve projdu seznam *points*, který obsahuje body křivky, a zkontroluji, zda se v blízkém okolí[14] kurzoru nenachází některý bod. Pokud ano, bude se pracovat s tímto bodem – stisknutí pravého myšítka způsobí vymazání bodu, levým myšítkem budeme vybraným bodem pohybovat. Pokud se v blízkosti žádný bod nenachází, vloží se na příslušné místo do seznamu nový bod.

Při hýbání bodem se volá metoda *recalculate()*, která zavolá metodu příslušející zvolenému typu interpolace křivky. Tato metoda využije zadané body pro interpolaci hodnot převodních funkcí. Při interpolaci přímkami jsou hodnoty mezi dvěma body počítány lineární interpolací, tedy ze vztahu $X = A + t * (B - A)$, kde A je počáteční bod úseku, B koncový bod a t parametr určující pozici výsledného bodu na daném úseku. Interpolace pomocí kubických křivek je zde spíše pro ukázku možnosti výběru typu interpolace, samotná implementace metody dává velice nepřesné výsledky.

[14] Blízké okolí je definováno konstantou *POINT_RADIUS*.

Při změně převodní funkce se v upravované vrstvě zavolá metoda *setModifiedCurves()*, která nastaví atribut *action* na *CURVES* a zajistí tím volání správné funkce (*applyCurves()*) při překreslování obrázku.

Při pohybu myši nad plátnem upravovaného obrázku se v histogramu vykresluje svislá čára v místě, které odpovídá hodnotě pixelu pod kurzorem. Při stisknutí klávesy CTRL a levého tlačítka myši se na toto místo do křivky vloží bod, pokud místo klávesy CTRL stiskneme SHIFT, vloží se bod do křivek všech kanálů. Vložení bodu probíhá v metodě *addPoint(int *values, bool allChannels)*, pole *values* obsahuje hodnoty pixelu v jednotlivých barevných kanálech a *allChannels* rozhoduje o tom, zda se má bod vložit do všech křivek či pouze do právě upravované. Vložení bodu probíhá podobně jako ve výše zmiňovaném případě, procházím pole bodů křivky a při nalezení správné pozice bod vložím do seznamu.

Stisknutí tlačítka *Reset* způsobí vymazání všech bodů upravované křivky až na první a poslední (bod [0, 0] a [255, 255]) a přepočítání převodních funkcí. Stisknutí tlačítka *Cancel* zresetuje všechny křivky a zavře nástroj, tlačítko *Ok* potvrdí úpravu obrázku a uloží ji do historie úprav.

3.3.6. Práh

Nástroj *Práh* je tvořen třídou *Threshold*. Poskytuje posuvník pro zadání hodnoty prahu, tento údaj je případně možné zadat i číselně do příslušného políčka. Dále umožňuje zapnutí či vypnutí náhledu a potvrzení barevné korekce.

Nástroj *Práh* slouží jako ukázka velmi jednoduché barevné korekce, na jeho příkladu ukážu v kapitole 3.4 způsob, jakým je možné do programu přidávat další nástroje pro barevné korekce.

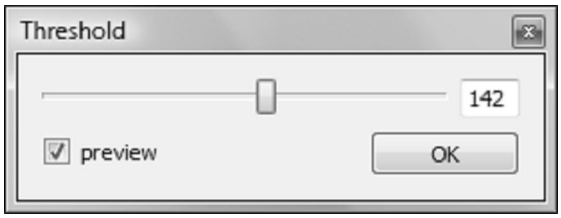
3.4. Přidání nástroje pro barevné korekce

Postup při přidání nového nástroje pro barevné korekce ukážu na příkladu jednoduchého nástroje *Práh* (třída *Threshold)*, ale jednotlivé kroky by měly být téměř stejné při vytváření libovolného nástroje.

3.4.1. Vytvoření GUI

Nejprve je nutné vytvořit grafické uživatelské rozhraní nového nástroje, k čemuž je možné s výhodou využít *Qt Designer*, nástroj pro návrh GUI šířený společně s knihovnou. Každý nástroj musí obsahovat checkbox pro zapnutí a vypnutí náhledu

a tlačítko pro potvrzení barevné korekce. Další prvky GUI závisí na potřebách konkrétního nástroje, v příkladě *Prahu* (obr. 3.9) se jedná o horizontální slider a políčko pro zadání přesné hodnoty, změny těchto prvků se projeví v nastavení prahu.



Obr. 3.9: Nástroj Práh

V konstruktoru je nutné nastavit atribut, který zaručí požadované chování nástroje, voláním metody *setWindowFlags(Qt::Tool)*. Po tomto nastavení bude nástroj správně reagovat na požadavky na minimalizaci či na obnovení hlavního okna programu.

3.4.2. Přidání nástroje do hlavního okna

Ve třídě *MainWindow* je nutné přidat atribut *Threshold *threshold* a metodu pro otevření nástroje *showThreshold()*. V té se metodou *show()* a *activateWindow()* zobrazí a aktivuje okno nástroje, metodou *setToolVisible()* se nastaví atribut třídy, který uchovává informaci o tom, zda je otevřený některý z nástrojů, dále se nástroji nastaví ukazatel na vrstvu, která se bude upravovat, a nakonec se zavolá metoda *disableTools()*, která zaručí, že nebude možné zapnout další nástroj či přepínat mezi obrázky, dokud úprava neskončí.

V GUI dále doplníme odpovídající položku do menu *Image* a do toolbaru, příslušný signál propojíme s vytvořenou metodou *showThreshold()*. Prvky GUI musí být při spuštění programu zablokované, odblokují se při otevření nového obrázku v metodě *restoreTools()* – aby program správně fungoval i s novým nástrojem, je nutné do metod *restoreTools()* a *disableTools()* doplnit nastavení blokování nástroje metodou *ui.actionThreshold->setEnabled()*.

3.4.3. Implementace signálů a slotů

Nástroj musí překrýt metody *closeEvent()*, *hideEvent()* a *showEvent()* a poskytovat signály *thresholdClosed()*, *thresholdEnabled(bool)* a *repaintNeeded()*. Signál *thresholdClosed()* vyvolaný v metodě *closeEvent()* je propojený se slotem *restoreTools()* třídy *MainWindow*, zaručí tedy odblokování nástrojů a opět umožní

přepínání mezi jednotlivými obrázky při vypnutí nástroje. *ThresholdEnabled(bool)* je vysílán v metodách *hideEvent()* a *showEvent(),* propojený je se slotem *setPreview(bool)* třídy *Canvas*, stará se o správné nastavení náhledu při skrytí nebo vyvolání nástroje. Signál *repaintCanvas()* propojený se slotem *repaintActiveCanvas()* třídy *MainWindow* je volán při změně nastavení nástroje a zaručí překreslení plátna upravovaného obrázku.

Dále je nutné propojit signál vyvolaný při stisknutí tlačítka pro potvrzení barevné korekce se slotem *saveHistory()* třídy *MainWindow,* který se postará o provedení barevné korekce na celém obrázku a uložení úpravy do historie. Tento signál je také potřeba propojit se slotem *close()* okna nástroje, aby se nástroj po uložení korekce zavřel.

Signál vyvolaný změnou stavu checkboxu pro náhled musíme propojit se slotem *setPreview(bool)* třídy *Canvas*, čímž zaručíme správné vykreslení obrázku s úpravami nebo bez nich.

3.4.4. Atributy nástroje

Nástroj musí obsahovat atributy *AbstractImage *image* (ukazatel na upravovaný obrázek) a dvourozměrné pole *int **transitFunction* (převodní pole pro každý kanál obrázku). Oba tyto atributy se nastavují v metodě *setImage()*. Převodní pole se získá z upravovaného obrázku, nové se vytvoří pouze pokud obrázek zatím žádné neobsahuje.

3.4.5. Funkčnost nástroje

Dále je nutné implementovat funkčnost konkrétního nástroje, tedy přepočet hodnot převodních polí pro jednotlivé kanály na základě parametrů nastavených v GUI. Přepočet se odehrává v metodě *recalculate()*, která je volána při každé změně nastavení parametrů nástroje a která vypočte nové hodnoty pole.

Po přepočtu je nutné upravované vrstvě nastavit atribut *bool modified*, čímž se zaručí přepočtení obrázku před vykreslením, pro použití správné funkce pro barevnou korekci *(applyThreshold())* je nutné vhodně nastavit atribut vrstvy *ModifyingAction action*. V případě *Prahu* jsou tyto atributy nastaveny metodou *setModifiedthreshold()*. Stejné metody lze využít pro všechny nástroje pracující s jasem obrázku, pro nástroje pracující s jednotlivými kanály je možné využít obdobné metody *setModifiedCurves()* a *applyCurves()*.

Při implementaci barevné korekce, pro kterou není použitelný ani jeden z výše zmíněných přístupů, musíme do výčtu *ModifyingAction* doplnit nový typ korekce a při

změně parametrů jej nastavit do *ModifyingAction action*. Dále musíme implementovat novou metodu pro požadovaný způsob manipulace s jednotlivými pixely obrázku, a její volání přidat do přepínače v metodě *applyAction()* třídy *AbstractImage*, například při implementaci *Prahu* se jednalo o přidání řádky *case THRESHOLD: applyThreshold(); break;*.

3.4.6. Další nastavení

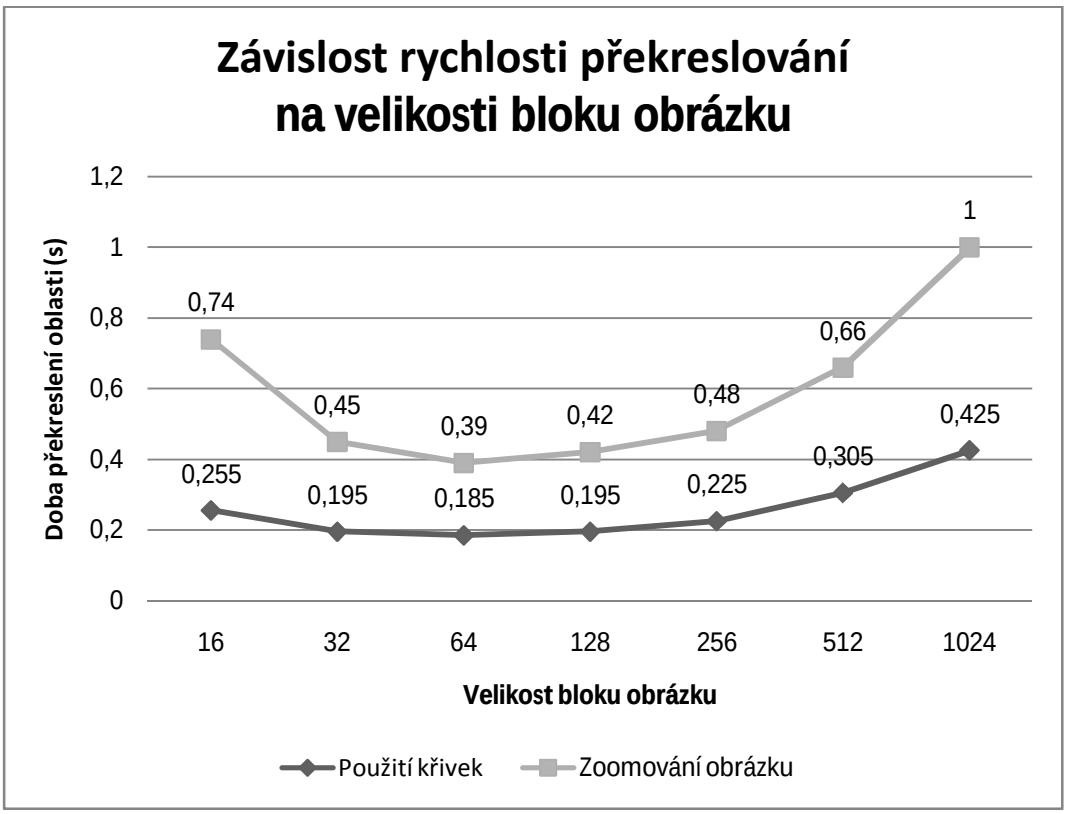
Po potvrzení úpravy by měl nástroj zařídit nastavení výchozích hodnot převodních polí, hodnota pole by měla být rovna indexu (vstupní hodnota se rovná výstupní hodnotě).

Posledním krokem, který je společný pro implementaci všech nástrojů, je smazání příslušného nástroje při vypnutí programu v metodě *closeEvent()* třídy *MainWindow*.

4. Testy a dosažené výsledky

4.1. Velikost bloku

Volba velikosti bloku obrázku má velký vliv na rychlost odezvy programu. V kapitole 3.1.3 jsem vysvětlila, proč je důležité volit velikost bloku jako mocninu čísla 2. Pro zjištění, jaká velikost bloku bude vykazovat nejlepší výsledky, jsem provedla následující test. V programu jsem otevřela obrázek a roztáhla jej přes celou plochu, konkrétně se jednalo o oblast 1661 x 904 pixelů. Poté jsem změřila průměrnou rychlost překreslení oblasti[15] v dvou různých situacích, a to při použití nástroje Křivky a při zoomování obrázku, výsledky jsou zpracovány v grafu na obr. 4.1.

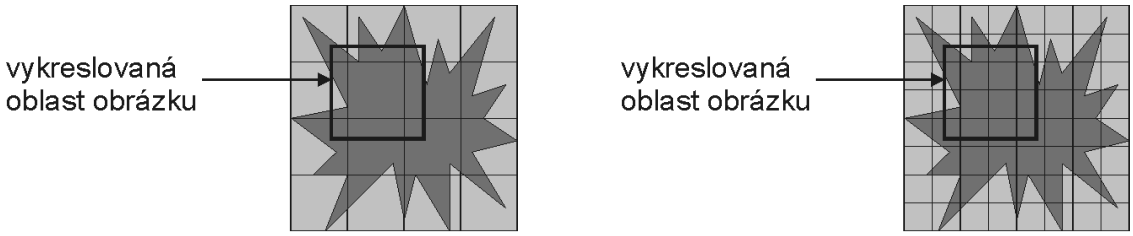


Obr. 4.1: Graf závislosti rychlosti překreslování na velikosti bloku obrázku

Při práci s obrázkem se požadovaná akce provede vždy na celém bloku, který alespoň částečně zasahuje do vykreslované oblasti. Při zoomování se tedy dopočítává i část bloků na okraji oblasti, které nutně nepotřebujeme, podobně při použití křivek se

[15] Všechny testy byly prováděny na počítači s touto konfigurací: Intel Core 2 Duo - Meron T7250 (Dual Core, 2,0GHz, 2x 2MB cache, 667MHz FSB), 2048 MB DDR2, NVIDIA GeForce 8600M GT 512MB VRAM

přepočítávají i hodnoty těchto přesahujících částí. Z toho vyplývá, že nevhodná volba velikosti bloku může vést k přepočítávání mnohonásobně větší oblasti, než jakou potřebujeme. Nejhorší možný případ je znázorněn na obrázku 4.2a. Vykreslovaná oblast je nepatrně větší než blok obrázku a zasahuje do devíti bloků, při překreslování obrázku je tedy nutné aktualizovat všechny tyto bloky, v krajním případě téměř devítinásobek potřebné oblasti.



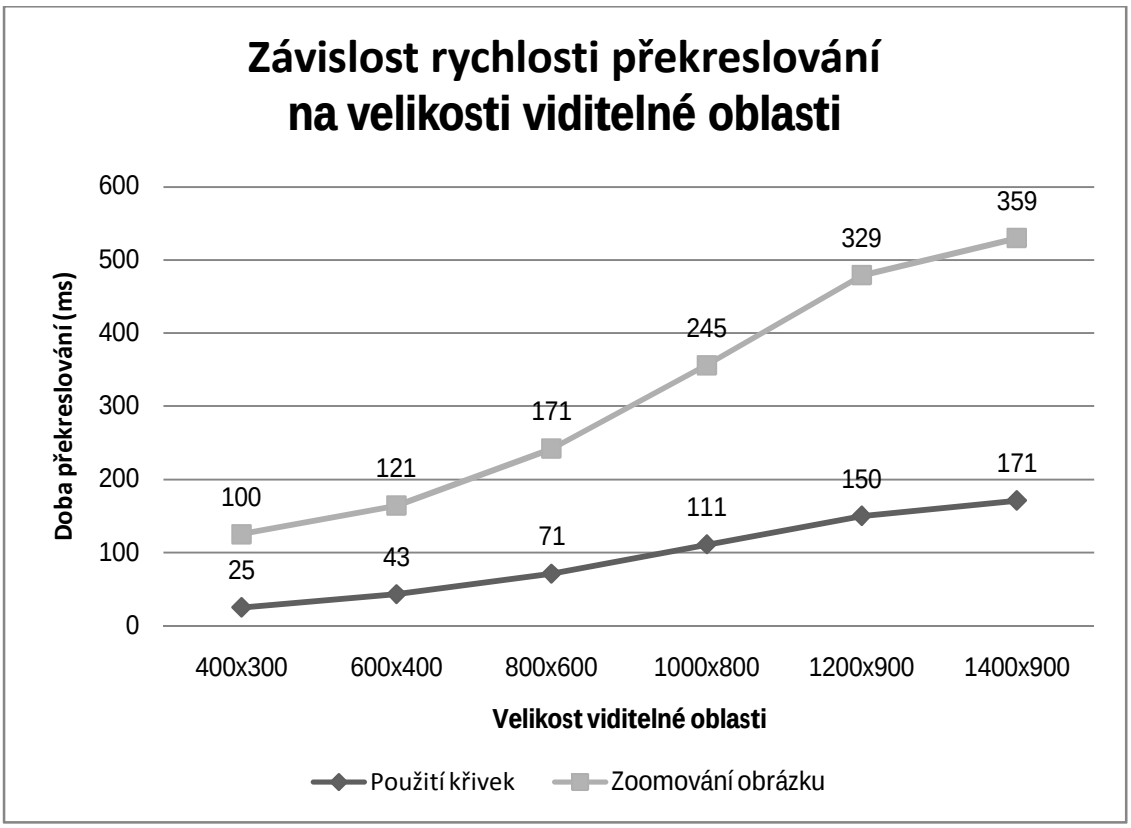
(a) Příliš velké bloky (b) Bloky poloviční velikosti

Obr. 4.2: Bloky potřebné pro vykreslení dané oblasti.

Zmenšením velikosti bloku klesá množství dat, která přepočítáváme navíc. Při zmenšení velikosti bloku na polovinu (viz obr. 4.2b), je nutné aktualizovat jen 4/9 oblasti z předchozího případu, v nejhorším případě je nutné přepočítat čtyřnásobek potřebné oblasti. Blok ale nemůžeme zmenšovat donekonečna – čím menší bude blok, tím více bloků bude potřeba při překreslování, a proto bude narůstat doba strávená voláním funkcí pracujících s bloky. Z obrázku 4.1 je vidět, že režie volání funkcí začne převažovat nad výhodami zmenšení bloku při velikosti bloku 32x32 pixelů. Nejlepší výsledky dává velikost 64x64 pixelů, rozdíly oproti výsledkům při velikosti bloku 128x128 pixelů jsou však zanedbatelné. Musíme vzít v úvahu i to, že sama matice ukazatelů na bloky zabírá místo v paměti (každý ukazatel zabírá 4 byty). Při načtení obrázku v rozlišení 3648x2736 potřebujeme pro uložení obrázku 638 bloků velikosti 128x128, nebo 2451 bloků velikosti 64x64, matice těchto rozměrů příliš místa nezaberou. Avšak při maximálním přiblížení obrázku (aplikace poskytuje maximální zoom 128x) už potřebujeme pro uložení obrázku matici 9 980 928 bloků velikosti 128x128, nebo 39 923 712 bloků velikosti 64x64, paměťové nároky budou tedy výrazně větší. Po vyhodnocení výsledků provedených testů jsem zvolila velikost bloku obrázku 128x128 pixelů.

4.2. Doba překreslení

Jak již bylo zmíněno výše, při překreslování se aktualizují pouze bloky obrázku spadající do zobrazované oblasti, doba odezvy programu je tudíž závislá na velikosti okna s obrázkem. Graf na obrázku 4.3 ukazuje průměrnou dobu (v milisekundách) potřebnou pro překreslení viditelné oblasti. Z grafu je vidět, že není příliš vhodné provádět barevné korekce na maximalizovaném obrázku, při upravování obrázku, z kterého vidíme pouze část, bude odezva výrazně rychlejší. Obvykle nepotřebujeme zobrazovat oblast větší než 800x600, při této velikosti probíhá překreslování obrázku dostatečně plynule.



Obr. 4.3: Graf závislosti rychlosti překreslování na velikosti viditelné oblasti

Pokud provádíme korekce na obrázku, který se skládá z více vrstev, překreslování potrvá déle. Na výsledný obrázek budou mít vliv všechny částečně průhledné vrstvy, nad kterými se nenachází žádná neprůhledná. Graf na obr. 4.4 ukazuje, jak se bude měnit rychlost překreslování obrázku složeného z jedné, ze dvou a ze tří vrstev. Při obrázku z více vrstev by se časy potřebné pro překreslování dále zvyšovaly, což by vedlo k pomalejší odezvě programu. Tomuto by mohlo být možné částečně zabránit při

dalším rozšiřování programu – úpravy probíhají vždy pouze na jedné vrstvě, bude tedy možné všechny vrstvy, které se nacházejí pod ní, předem sloučit do pomocného obrázku. Při vytváření kompozitního obrázku pak budeme moci využít předpřipravený obrázek nižších vrstev a dosáhnout tím rychlejší odezvy.

Obr. 4.4: Graf závislosti rychlosti překreslování na velikosti viditelné oblasti a počtu vrstev

Doba na provedení barevné korekce a překreslení obrázku se při kompozitu z více vrstev zvyšuje, proto je výhodné při barevných korekcích zobrazovat pouze ty vrstvy, které k úpravě potřebujeme. Vrstvy, u kterých v nástroji *Layers* nastavíme neviditelnost, budou při vykreslování přeskočeny a nebudou jej tedy zpomalovat.

Nastavením vhodné velikosti viditelné oblasti obrázku a vypnutím viditelnosti vrstev, které nejsou pro prováděnou barevnou korekci nutné, lze dosáhnout výborné doby odezvy programu na změny nastavení nástroje pro barevnou korekci.

4.3. Spotřeba paměti

Program má velké paměťové nároky. Při práci s obrázkem udržuje zoomovací pyramidy v původním barevném prostoru i v barevném prostoru monitoru. Pyramidy obsahují pouze bloky, které jsme využili při vykreslování, čímž se paměťové nároky omezí. Pro další úsporu paměti při ukládání úpravy do historie z obrázku mažu všechna nepotřebná data, tedy všechny úrovně pyramid vyjma zoomu 100%, smazané úrovně mohu opět v případě potřeby dopočítat.

I přes tato opatření se může stát, že programu začne docházet paměť, obzvlášť při práci s obrázkem ve velkém rozlišení, proto jsem v programu omezila délku historie (počet úprav, které je možné vrátit zpět). Vhodnějším řešením by bylo, kdyby program za běhu zjišťoval, kolik má k dispozici volné paměti a délku historie omezil až ve chvíli, kdy paměť dochází. Bohužel se mi nepodařilo najít platformně nezávislý způsob, jak tohoto dosáhnout, proto bude při dalším rozšiřování programu pravděpodobně nutné tento problém vyřešit pomocí podmíněného překladu. Jiným možným řešením by bylo v programu pevně nastavovat, kolik paměti smí používat, a začít mazat až v případě nutnosti.

4.4. Nástroje pro barevné korekce

Implementované nástroje (především Křivky) jsem důkladně otestovala na sadě fotografií, některé dosažené výsledky jsem zařadila do přílohy A. Nástroje se chovají dle předpokladu, s jejich využitím je možné dosáhnout podobných výsledků jako s ekvivalentními nástroji profesionálních programů pro barevné korekce.

5. Závěr

Barevné korekce jsou velice důležité pro profesionální fotografy, mohou však být užitečné i pro běžného uživatele, pro kterého jsou profesionální aplikace často nedostupné kvůli své vysoké ceně. Hlavním cílem této práce bylo implementovat přenositelnou aplikaci, která bude šířena jako open source a která bude poskytovat základní nástroje pro barevné korekce, stejně jako obecnou funkcionalitu potřebnou při práci s obrázkem. Tento cíl byl splněn, vytvořená aplikace Dikobraz umožňuje prohlížení fotografií v různém zvětšení a výřezu, pracuje s vrstvami a historií úprav a obsahuje nástroj pro přesnou práci s hodnotami pixelů. Dále poskytuje požadované nástroje pro barevné korekce, konkrétně se jedná o Práh, Křivky a nástroj pro násobení barevných kanálů konstantní hodnotou, jejich funkčnost je srovnatelná s profesionálními nástroji pro barevné korekce.

Během vývoje aplikace několikrát nastala situace, kdy se použitý způsob řešení problému ukázal být nevhodný pro svou pomalost, a bylo nutné přistoupit k jinému, časově méně náročnému řešení. Výsledná aplikace provádí barevné korekce velice rychle, přesto je nepochybně možné použité metody dále zdokonalit a dosáhnout ještě lepších výsledků. Snaha o dosažení co nejlepších výsledků v rychlosti byla v některých případech kompenzována vyšší paměťovou náročností, kterou považuji za nejslabší místo programu.

Aplikace byla již od začátku vytvářena s důrazem na umožnění dalšího rozšiřování, proto k ní byla vytvořena důsledná dokumentace datových struktur, infrastruktury a zdrojového kódu. Při dalším rozšiřování aplikace je možné zaměřit se na zvýšení rychlosti aplikace a snížení její paměťové náročnosti. Jiným možným směrem vývoje projektu je implementace dalších nástrojů pro barevné korekce či dodání podpory práce s obrázky v dalších barevných prostorech.

Přehled zkratek

CIE	Commission internationale de l'éclairage (Mezinárodní komise pro osvětlení)
GNU	rekurzivní zkratka pro GNU's Not Unix, projekt zaměřený na svobodný software
GUI	graphical user interface (grafické uživatelské rozhraní)
FLTK	Fast, Light Toolkit – knihovna pro vytváření GUI
GTK+	The GIMP Toolkit – knihovna pro vytváření GUI
GIMP	GNU Image Manipulation Program, program pro upravování obrázků
Qt	knihovna pro vytváření GUI
LPGL	Lesser General Public License, licence svobodného softwaru
GPL	General Public Licence, licence svobodného softwaru
BSD	Berkeley Software Distribution, permisivní licence svobodného softwaru
MIT	Massachusetts Institute of Technology, MIT License je permisivní licence svobodného softwaru

Literatura

[1] *Gimp příručka: Barvy a jejich nástroje* [online]. [cit. 2009-04-16]. URL: < http://www.gimp.kvalitne.cz/barev_nastroj.htm>

[2] *Gimp: ÚPRAVY FOTOGRAFIÍ* [online]. [cit. 2009-04-16]. URL: < http://www.linuxsoft.cz/article.php?id_article=376>

[3] KOLINGEROVÁ, I. *Podpora počítačové kreativity* [online]. [cit. 2009-04-17]. URL: < http://iason.zcu.cz/~kolinger/vyukaZCU.html#POKR>

[4] MARGULIS, D. *Professional Photoshop: The Classic Guide to Color Correction.* 5th edition. Berkeley (California): Peachpit Press, 2007. ISBN 0-321-44017-X.

[5] PŘIKRYL, P. – BRANDNER, M. *Numerické metody II*. 1. vydání. Plzeň: Západočeská univerzita, 2000. ISBN 80-7082-699-1.

[6] LOBAZ, P. *Barevná úprava fotografií* [přednáška]. Plzeň: Západočeská univerzita, 2007.

[7] *Photoshop pro začátečníky* [online]. [cit. 2009-04-15]. URL: < http://www.owebu.cz/photoshop/>

[8] *Photoshop Tutorials - Adobe Photoshop Tutorials* [online]. [cit. 2009-04-15]. URL: < http://www.photoshopessentials.com/>

[9] PASCAL, D. *A review of RGB Colorspaces.* Montreal (Canada): The BabelColor Company, 2003. URL:<http://www.babelcolor.com/download/A%20review%20of%20RGB%20color%20spaces.pdf >

[10] *Základy počítačové grafiky* [online]. [cit. 2009-04-18]. URL: < http://herakles.zcu.cz/education/zpg/cviceni.php>

[11] SKALA, V. *Světlo, barvy a barevné systémy v počítačové grafice*. 1. vydání. Praha: Academia, 1993. ISBN 80-200-0463-7. URL: < http://herakles.zcu.cz/education/zpg/link.php>

[12] *Wikipedia, the free encyclopedia.* [online]. URL: < http://www.wikipedia.org>

[13] PRATA, S. *Mistrovství v C++*. 1. vydání. Praha: Computer Press, 2001. ISBN 80-7226-339-0.

[14] BLANCHETTE, J. – SUMMERFIELD, M. *C++ GUI Programming with Qt4.* 1st printing. Stoughton (Massachusetts): Courier, 2006. ISBN 0-13-187249-4.

[15] *Iron Spider: web design and resource center* [online]. [cit. 2009-05-06] URL <http://www.ironspider.ca/freetools/grfxeditors.htm>

Příloha A

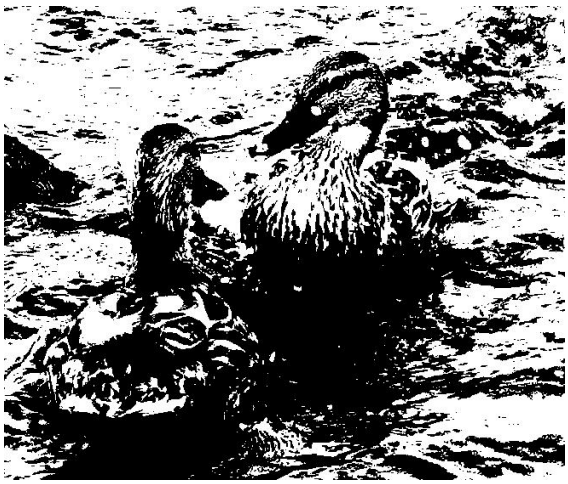
Ukázky použití nástrojů pro barevné korekce

Popisek obrázku vždy obsahuje informaci o tom, zda se jedná o původní fotografii, nebo s pomocí kterého nástroje obrázek vznikl.

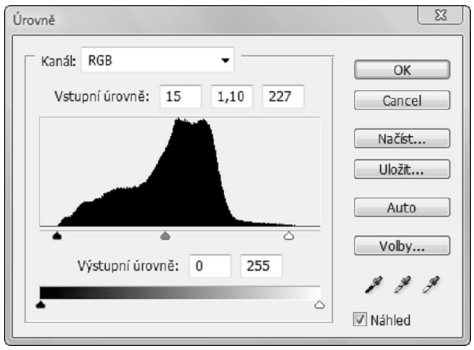
(a) Originál

(b) Jas-kontrast

(c) Úrovně



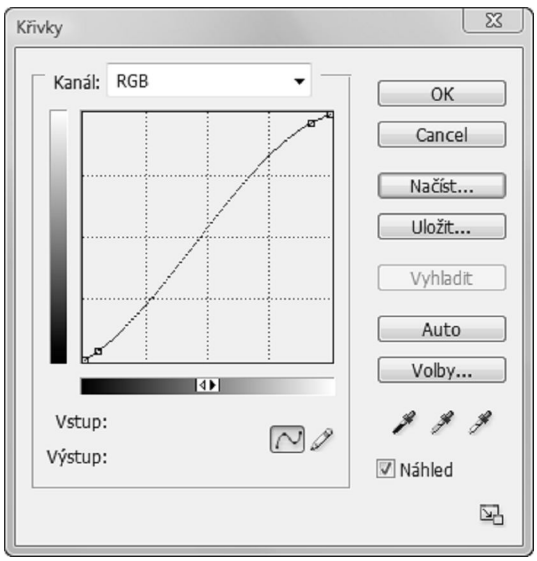
(d) Práh



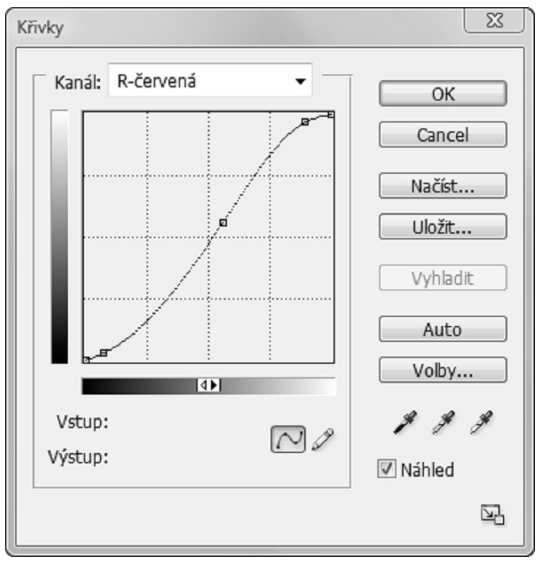
(e) Parametry nástroje Úrovně pro vytvoření obr. A.1c

Obr. A.1: (a) Originál. (b) Jas-kontrast (Jas 7, kontrast 30). (c) Úrovně (RGB: 15, 1.1, 227; R: 24, 0.91, 231; G: 17, 0.89, 230; B: 12, 0.88, 220). (d) Práh (hodnota prahu 128). (e) Screenshot nástroje Úrovně s parametry použitými pro vytvoření (c) – kanál RGB.

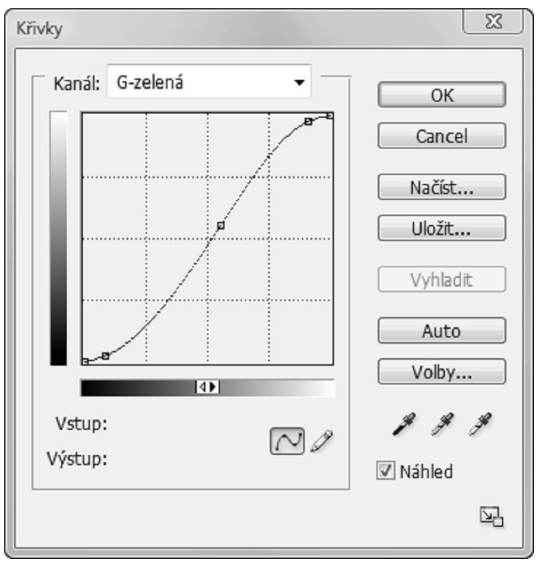
(a) Křivky



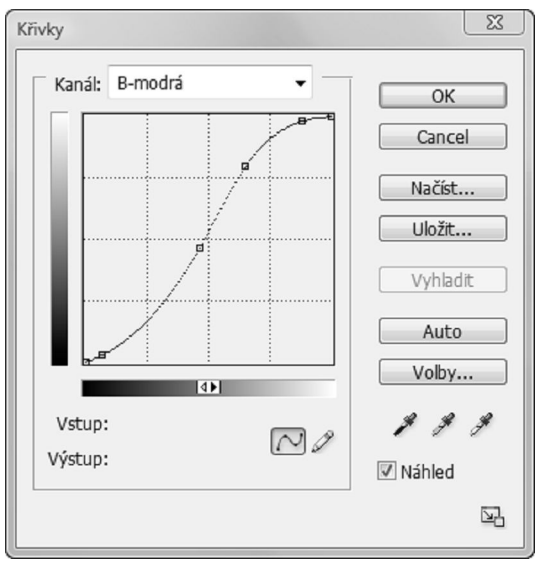
(b) Kanál RGB



(c) Kanál R



(d) Kanál G



(e) Kanál B

Obr. A.2: (a) Obrázek upravený pomocí nástroje Křivky. (b) Křivka kanálu RGB. (c) Křivka kanálu R. (d) Křivka kanálu G. (e) Křivka kanálu B.

(a) Originál (b) Odstín-sytost

Obr. A.3: (a) Originál. (b) Odstín-sytost (žluté barvy – H -28, S 0, V 0).

(a) Originál (b) Selektivní barva

Obr.A.4: (a) Originál. (b) Selektivní barva (zelené – C +100%, M -100%, Y +100%, K +100%).

(a) Originál (b) Použít obraz

Obr.A.5: (a) Originál. (b) Použít obraz (Kanál RGB, režim Překrýt, krytí 70%).

(a) Originál

(b) Převod na stupně šedi (c) Míchání kanálů

Obr. A.6: (a) Originál. (b) Převod na stupně šedi. (c) Míchání kanálů (monochromatický režim – R -20%, G 100%, B 40%)

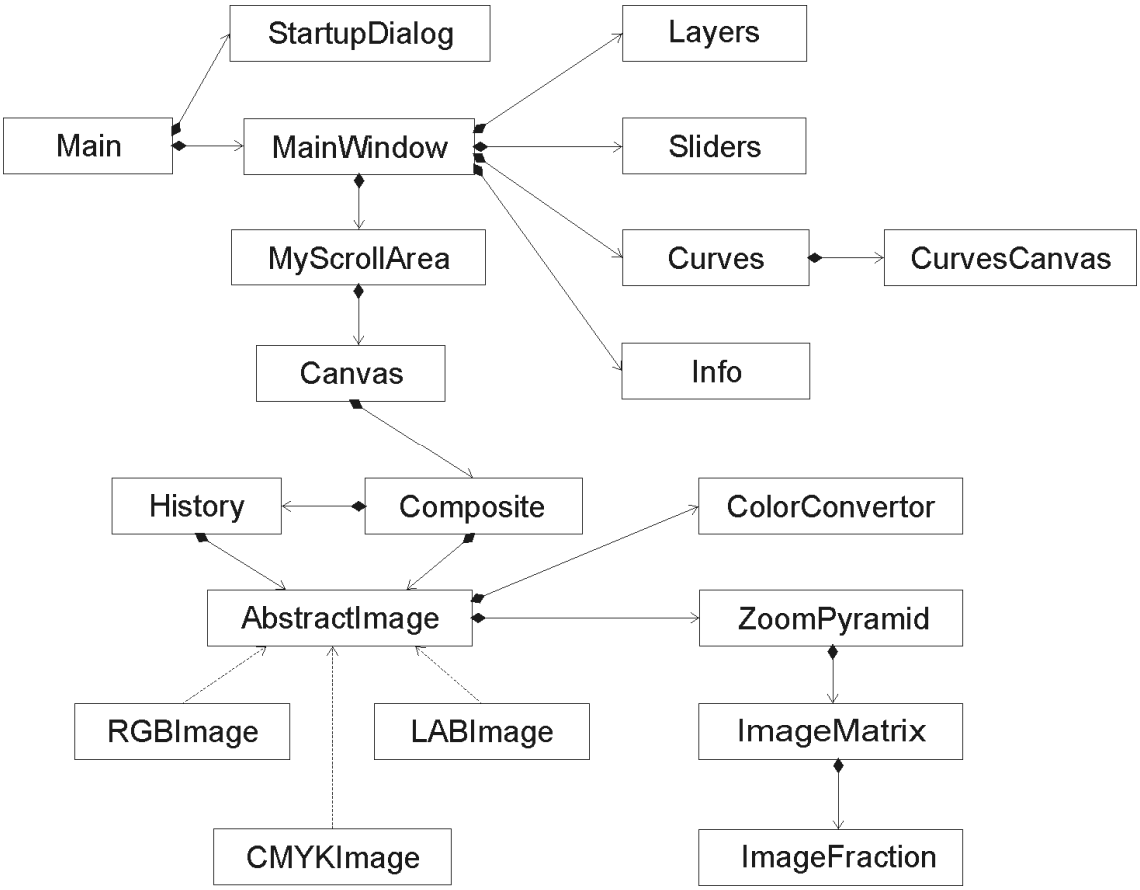
Ukázky výsledků aplikace Dikobraz

Obrázky vlevo jsou originální fotografií, vpravo jsou ukázány výsledky dosažené pomocí nástrojů aplikace Dikobraz.

Obr. A.7: Použití nástroje Práh

Obr. A.8: Použití nástroje Křivky

Další obrázky



Obr. A.9: Znázornění relací mezi třídami aplikace Dikobraz

Obr. A.10: Uvítací obrazovka aplikace Dikobraz

Příloha B

Postup při překladu

Před překladem programu je nutné nainstalovat překladač jazyka C++ MinGW a knihovnu Qt pro C++. Překladač MinGW je dostupný na adrese *http://www.mingw.org/*, případně *http://sourceforge.net/project/showfiles.php?group_id=2435*, není však nutné jej instalovat odděleně, instalátor knihovny Qt se o správnou instalaci dokáže postarat sám (pokud je počítač v době instalace připojený k internetu). Knihovna Qt je k dispozici na *http://www.qtsoftware.com/downloads*. Po úspěšném nainstalování je nutné do proměnné PATH přidat příslušné cesty, např. *C:/Qt/4.4.0/bin* a *C:/MinGW/bin*[16]. Pokud je vše správně nainstalováno a nastaveno, mělo by nyní v pracovním adresáři fungovat volání *qmake* a *mingw32-make*.

Pro přeložení programu jsou potřebné soubory *Makefile, BP.pro, main.cpp* a adresáře *icons, include* a *src*. Nejprve je nutné v adresáři se souborem *Makefile* zavolat *qmake*, který vygeneruje soubory nutné pro překlad, následným voláním *mingw32-make* se spustí samotný překlad. Pokud byl překlad úspěšný, objeví se v adresáři *debug* spustitelný soubor *Dikobraz.exe*.

Uživatelský manuál

Po spuštění programu ze souboru *Dikobraz.exe* se zobrazí hlavní okno (obr. B.1), které obsahuje menu a toolbary s ovládacími prvky.

Obr. B.1: Hlavní okno programu

[16] Cesta k nainstalovaným nástrojům, přesné umístění zadáváte při instalaci.

Otevření nového obrázku

Po zvolení položky menu *File–>Open*, příslušné ikony v toolbaru (první zleva) nebo po stisknutí kláves *CTRL+O* se objeví dialogové okno pro výběr obrázku ve formátu JPG, PNG nebo BMP. Při prvním spuštění aplikace dialog zobrazí obsah adresáře, kde se program nachází, při dalších spuštěních se objeví v adresáři, z kterého byl načítán obrázek naposledy. Po vybrání obrázku a potvrzení tlačítkem *Open* se vytvoří nové podokno, které bude obsahovat zvolený obrázek (viz obr. B.2).

Titulek okna bude obsahovat název otevřeného souboru, aktuální rozlišení obrázku a hodnotu zoomu. Roztáhnout okno přes celou plochu programu je možné kliknutím na ikonu pro maximalizaci okna, dvojklikem na záhlaví nebo tažením za okraj okna, zmenšení okna probíhá obdobně. K zavření obrázku dojde po stisknutí křížku v záhlaví nebo po zvolení položky menu *Window–>Close*.

Obr. B.2: Hlavní okno programu s otevřeným obrázkem

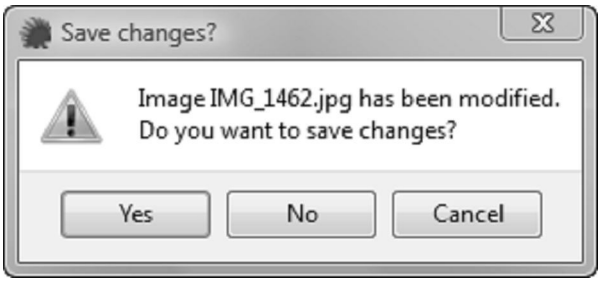
Správa současně otevřených obrázků

Pokud je v programu otevřeno více obrázku najednou, je možné ovlivňovat jejich zobrazení pomocí položek menu *Window*. Položky *Tile* a *Cascade* slouží k uspořádání obrázků na ploše programu, položky *Next* a *Previous* přepínají mezi obrázky v pořadí, ve kterém byly otevřeny. V menu *Window* se také zobrazuje seznam aktuálně otevřených obrázků, zvolením některého z nich se aktivuje příslušné okno. Zvolením položky *Close All* se zavřou všechny obrázky.

Uložení obrázku

Po zvolení položky menu *File–>Save,* ikony v toolbaru (ikona diskety) nebo po stisknutí kláves *CTRL+S* se otevře dialogové okno pro uložení obrázku do formátu JPG, PNG nebo BMP, po zvolení adresáře a názvu souboru a po potvrzení tlačítkem *Save* se obrázek uloží. Pokud se obrázek skládá z více vrstev, před uložením se nejprve vytvoří kompozitní obrázek se zohledněním průhlednosti a teprve ten se uloží do zvoleného souboru.

Při zavírání okna s obrázkem, na kterém byly provedeny změny, se zobrazí upozornění o neuložených změnách, viz obr. B.3, a uživatel je vyzván k jejich uložení. Pokud zvolí tlačítko *Yes,* zobrazí se stejné dialogové okno jako při standardním ukládání, ve kterém opět uživatel zvolí adresář a název ukládaného souboru. Při volbě *No* se obrázek zavře bez uložení, provedené změny se zahodí. Tlačítkem *Cancel* se zruší zavírání okna, změny zůstanou neuloženy.



Obr. B.3: Dotaz na uložení změn v obrázku

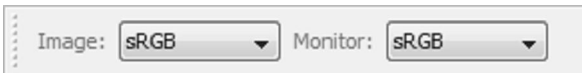
Manipulace s obrázkem

Pokud se obrázek v daném přiblížení nevejde do okna celý, zobrazí se po stranách posuvníky pro posun zobrazované části obrázku. Posouvat obrázek je možné i myší – stisknutím mezerníku se zapne posuv myší, výchozí kurzor změní na kurzor ve tvaru ruky. Obrázkem je pak možné posouvat pohybem myši se stisknutím myšítkem. Opětovným stisknutím mezerníku se pohyb myší vypne a kurzor se přepne zpět.

Obrázek je možné přiblížit zvolením položky menu *View–>Zoom In,* příslušné ikony v toolbaru nebo stiskem klávesy +, oddálení probíhá obdobně zvolením položky *View–>Zoom Out,* ikony v toolbaru nebo klávesou -. Při zoomování obrázku je zaručeno, že oblast obrázku pod kurzorem myši zůstane viditelná i po provedení požadované akce.

Nastavení barevného prostoru

Program umožňuje nastavit, v jakém barevném prostoru jsou data načteného obrázku a jaký je barevný prostor monitoru. Pro nastavení barevného prostoru obrázku jsou určeny položky v menu *View–>Image Colorspace,* případně položky v toolbaru s označením *Image* (viz obr. B.4); barevný prostor monitoru se nastavuje položkami menu *View–>Monitor Colorspace* nebo položkami v toolbaru s označením *Monitor* (viz obr. B.4).



Obr. B.4: Položky toolbaru pro nastavení barevných prostorů

Pokud je otevřeno více obrázků najednou, je možné, aby měl každý své nastavení. Při přepínání mezi obrázky se pak bude v GUI programu aktualizovat nastavení barevných prostorů podle nastavení konkrétního obrázku.

Odečítání přesných hodnot pixelu

K odečítání přesných hodnot pixelu slouží nástroj *Info* (obr. B.5a, B.5b), který se vyvolá zvolením položky menu *Image–>Info*, příslušnou položkou v toolbaru nebo klávesovou zkratkou *CTRL+P*. V levé polovině okna jsou zobrazovány údaje o pixelu v barevném prostoru obrázku, v pravé části je možné zobrazit ekvivalentní hodnoty v *RGB, CMYKu* nebo *HSV*. Pro vybrání barevného prostoru stačí kliknout pravým myšítkem na okno nástroje a následně v kontextovém menu vybrat požadovaný prostor.

(a) Vzhled nástroje při prohlížení (b) Vzhled nástroje při provádění úprav

Obr. B.5: Nástroj Informace

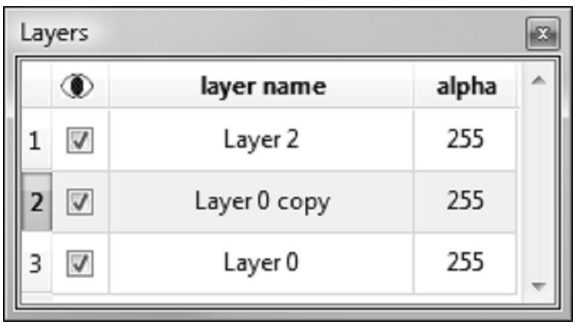
Při prohlížení obrázku bez otevřeného nástroje pro barevné korekce zobrazuje nástroj hodnoty pixelu pod kurzorem myši, viz obr. B.5a. Při probíhající úpravě obrázku

a zapnutém náhledu změn nástroj ukazuje původní hodnotu pixelu i hodnotu po provedení dané barevné korekce oddělené lomítkem (viz obr. B.5b), na levé straně lomítka je původní hodnota a vpravo hodnota pixelu po provedení korekce.

Nástroj umožňuje odečítání přesné hodnoty pixelu pod kurzorem nebo získání průměrné hodnoty na 3x3, případně 5x5 okolí. K přepínání mezi těmito možnostmi slouží položky menu *Image–>Sample Size,* položky v toolbaru označené *Sample Size* nebo klávesové zkratky *CTRL+1, CTRL+2* a *CTRL+3.*

Manipulace s vrstvami

Pro práci s vrstvami slouží okno *Layers* (obr. B.6), které se otevře po zvolení položky menu *Image–>Layers,* položky *Layers* v toolbaru nebo klávesovou zkratkou *CTRL+L.* V okně se zobrazuje seznam všech vrstev daného obrázku, první vrstva odshora je nejvyšší vrstvou kompozitu, další vrstvy následují v pořadí, v jakém jsou uvedeny v nástroji. Nástroj umožňuje změnu viditelnosti vrstvy (druhý sloupec s ikonou oka), jejího jména a její průhlednosti.



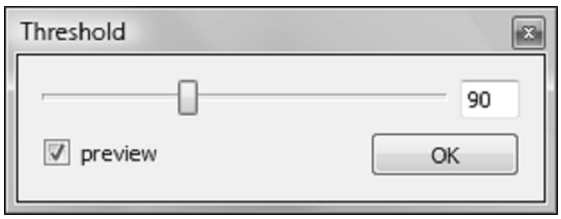
Obr. B.6: Nástroj pro manipulaci s vstvami

Po kliknutí pravým tlačítkem myši kdekoli v okně se zobrazí kontextové menu s položkami pro vytvoření nové vrstvy, zkopírování zvolené vrstvy či její vymazání. Při vytváření nové vrstvy se zobrazí dialog pro výběr barvy, kterou bude nová vrstva vyplněna. Pořadí vrstev je možné měnit přetažením myší.

Ovládání nástroje Práh

Nástroj *Práh* (obr. B.7) se otevře po zvolení položky menu *Image–>Threshold,* položky *Threshold* v toolbaru nebo stisknutím kláves *CTRL+T.* K nastavení hodnoty prahu slouží posuvník a pole, do kterého je možné zapsat přesnou číselnou hodnotu. Do tohoto pole lze zadat pouze hodnoty v rozsahu 0 až 255, jiné hodnoty nástroj nepovolí.

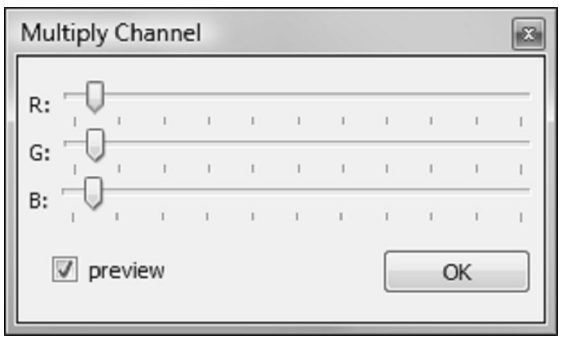
Pokud je zapnutý náhled (preview), změna prahu se okamžitě projeví v upravovaném obrázku. Stisknutím tlačítka *OK* se uloží provedené změny a okno nástroje se zavře.



Obr. B.7: Nástroj Práh

Ovládání nástroje Násobení kanálu

Násobení kanálu (obr. B.8) se otevře po zvolení položky menu *Image–>Multiply Channel*, položky *Multiply Channel* v toolbaru nebo stisknutím kláves *CTRL+M*. Změnou polohy posuvníku se nastavuje konstanta, kterou bude vynásobena příslušná barevná složka každého pixelu obrázku. Při zapnutém náhledu se bude obrázek aktualizovat po každé změně nastavení. Stisknutím tlačítka *OK* se uloží provedené změny a okno nástroje se zavře.



Obr. B.8: Nástroj Násobení kanálu

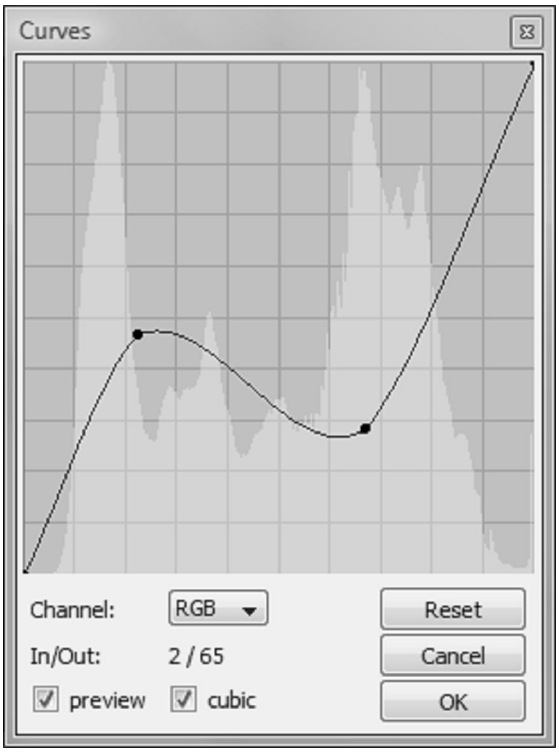
Ovládání nástroje křivky

Nástroj *Křivky* (obr. B.9) se otevře po zvolení položky menu *Image–>Curves*, položky *Curves* v toolbaru nebo stisknutím kláves *CTRL+K*. Nejprve je nutné v příslušném comboboxu zvolit kanál obrázku, který se má upravovat (upravovaný kanál je také možno změnit pomocí klávesových zkratek – *0* způsobí přepnutí na kompozitní kanál, *1* na první barevný kanál, atd.). V okně se vykreslí histogram zvoleného barevného kanálu, který může být užitečný při provádění korekce.

Kliknutím levým myšítkem do oblasti s histogramem se na zvolené místo do křivky vloží nový bod, při podržení stisknutého tlačítka je možné jím pohybovat. Při kliknutí levým myšítkem na existující bod se nový bod nevkládá, ale opět je možné s bodem pohybovat, kliknutí na bod pravým myšítkem jej z křivky vymaže.

Při pohybu myší nad upravovaným obrázkem se v nástroji *Křivky* v oblasti histogramu vykresluje svislá čára, označující polohu pixelu pod kurzorem. Při stisknutí klávesy *CTRL* a kliknutí levým myšítkem se na toto místo do křivky zvoleného kanálu vloží nový bod, při stisknutí *SHIFT* a kliknutí se bod vloží na příslušné místo do křivek všech kanálů obrázku.

Stejně jako u výše uvedených nástrojů se změny nastavení okamžitě projeví v upravovaném obrázku v případě, že je zapnutý náhled. Checkbox *cubic* slouží pro přepnutí způsobu interpolace křivky, křivku je možné interpolovat lineárními úseky nebo kubickými křivkami. Stisknutí tlačítka *Reset* způsobí vymazání všech vložených bodů křivky, tlačítkem *Cancel* se zruší změny a nástroj se uzavře, při stisknutí tlačítka *OK* se před zavřením nástroje změny uloží.



Obr. B.9: Nástroj Křivky