

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

Bakalářská práce

Editace perokreseb

Abstrakt

Bakalářská práce se zabývá tvorbou pluginu pro aplikaci Adobe Photoshop, který bude implementovat algoritmus pro editaci perokreseb, ve vývojovém prostředí Microsoft Visual C++ s použitím Adobe Photoshop SDK 6.0.

Abstract

Bachelor thesis deals with the creation of plug-in for Adobe Photoshop application that shall implement the Black-and-white drawing adjustment in the developing environment Microsoft Visual C++ using Adobe Photoshop SDK 6.0.

Obsah

1. Úvod.....	- 5 -
2. Algoritmus zesilování a ztenčování čar	- 7 -
3. Zesílení / ztenčení o celé body	- 8 -
3.1. Způsoby definování okolí bodu	- 8 -
3.1.1. „Přirozený“ způsob se čtvercovou maticí okolí	- 8 -
3.1.2. „Přirozený“ způsob s kružnicovým výběrem okolí.....	- 9 -
3.1.3. Dopředné zjišťování okolí	- 9 -
3.1.4. Zpětné zjišťování okolí.....	- 9 -
3.1.5. „Dolní“ způsob definování okolí.....	- 10 -
3.1.6. „Horní“ způsob definování okolí.....	- 10 -
3.2. Zhodnocení jednotlivých způsobů definování okolí.....	- 10 -
3.3. Urychlení algoritmu zesílení o celé body	- 11 -
4. Zesilování čar o desetiny (setiny) bodu	- 14 -
4.1. Spočítání nové hodnoty pixelu bez nutnosti vyplnění celé matice	- 16 -
4.2. Zvýšení efektivity algoritmu zesilování o desetiny (setiny) bodu	- 17 -
5. Detekce pozadí (pozitivní x negativní zesilování čar)	- 21 -
6. Známé chyby algoritmu	- 24 -
7. Příprava na tisk.....	- 27 -
8. Plugin pro Adobe Photoshop a tvorba dialogu	- 28 -
8.1. Plugin.....	- 28 -
8.2. Dialog.....	- 30 -
9. Uživatelská příručka.....	- 30 -
10. Závěr.....	- 32 -
Použitá literatura.....	- 33 -

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

V Plzni, dne 5.května 2008

.....

Pavel Bžoch

1. Úvod

Zadáme-li do internetového vyhledávače heslo „perokresby v tisku“ budeme překvapeni, kolik publikací obsahuje perokresby a kolik lidí se zabývá úpravou těchto kreseb. Ani my si nedovedeme představit jakoukoli publikaci, učebnici, odbornou knihu, články a beletrii bez ilustrací, doprovodných kreseb, grafických doplňků. Ilustrační a grafické prvky vložené do textu se nemalou měrou podílejí na výsledném celkovém dojmu dokumentu.

Úkolem zadané práce „Editace perokreseb“ je ukázat, jak lze počítačovou technikou upravit nejen psaný text, projekci, ale i ilustraci dokumentů. Mnoho dokumentů se tak stává kvalitnějšími. Přes znalost a ovládnání postupů výpočetní techniky pro vytváření kreseb se často vracíme ke klasické kresbě. Je originální, přesto i zde je potřeba editace. Po její aplikaci však problémy a starosti uživatele nekončí. Důležité je perokresby pro potřeby DTP dále upravit, aby byly vhodné pro použití v tisku.

Objasnění důležitých pojmů použitých v textu:

Editace perokresby: Příprava, zpracování a úprava perokresby k tisku.

Perokresba, vývoj perokresby: Jde o kresebný záznam perem. Jako výtvarný druh vznikla v raném středověku, patří ke klasickým kresebným technikám. Výrazovým prostředkem je čára různé síly kreslená rákosovým perem, brkem, od 19. století ocelovým brkem pomocí inkoustu.

DTP: Kompletní zpracování předloh, které jsou určeny pro tisk.

Perokresba v DTP: Jedná se o obrázek, ve kterém je pixel obrázku zobrazen jedním bitem. Tento obrázek se vždy tiskne v maximálním rozlišení tiskárny (typicky 1200dpi). Při tisku se nepoužívá rastrování.

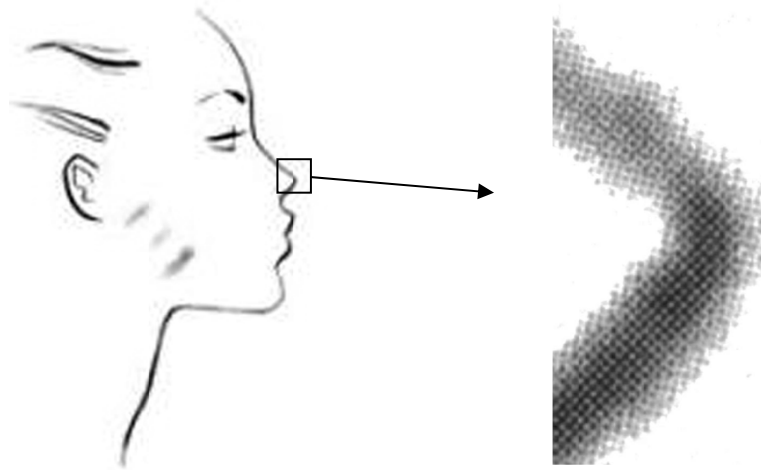
Ukázky perokreseb a rasterizace jsou vidět v následujících obrázcích (obr. 1.1 – 1.4)



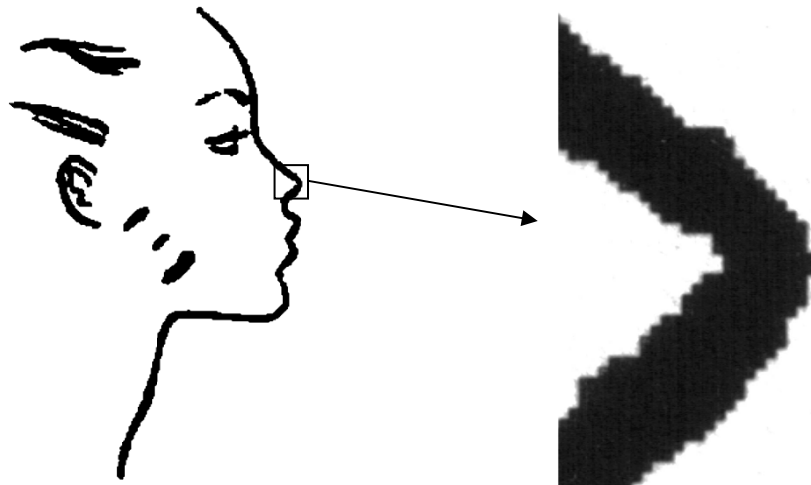
Obr. 1.1 Perokresba ve stupních šedi



Obr. 1.2 Perokresba jako bitová mapa (1pixel / 1bit)



Obr. 1.3 Scan tisku šedotónové perokresby s ukázkou rasterizace



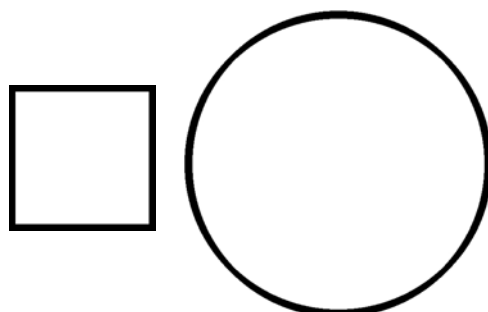
Obr. 1.4 Scan tisku perokresby jako bitové mapy bez rasterizace

V následujících odstavcích bude popsán algoritmus, jehož pomocí lze perokresbu připravit pro tisk. Tento algoritmus zahrnuje možnosti zesilování a ztenčování čar. Algoritmus

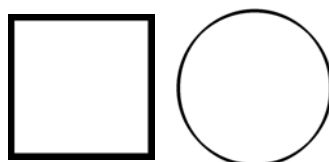
bude implementován jako zásuvný modul pro Adobe Photoshop. Výhodou této implementace je, že nemusíme uvažovat o různých formátech vstupních souborů. Photoshop nám vždy poskytne již načtený obrázek, se kterým můžeme dále pracovat. Nevýhodu spatřuji v nutnosti nastudování rozhraní Photoshopu pro zásuvné moduly.

2. Algoritmus zesilování a ztenčování čar

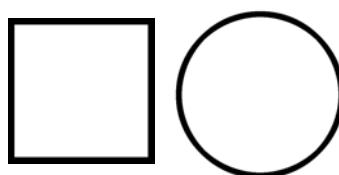
Prvotní otázka by mohla znít: „Proč vlastně zesilovat (nebo ztenčovat) čáry v perokresbě?“ Při přípravě tisku je potřeba „velkou“ perokresbu zmenšit (méně často zvětšit „malou“ perokresbu). Grafik by ovšem rád zachoval tloušťku čáry i po zmenšení (zvětšení) obrázku (viz obr. 2.1 – 2.3).



Obr. 2.1 „Perokresby“, které je potřeba zobrazit (vytisknout) stejně velké



Obr. 2.2 Při zmenšení došlo i k zúžení čar v kružnici



Obr. 2.2 Zesílení čáry v kružnici

Pro tyto potřeby je v programu použit algoritmus, který nejprve zesílí / ztenčí čáry o celé body a následně provede zesílení / ztenčení o desetiny (setiny) bodu. Jak samotné zesílení / ztenčení funguje, bude popsáno v následujících odstavcích.

3. Zesílení / ztenčení o celé body

Jako algoritmus pro zesílení / ztenčení čar o celé body byl vybrán algoritmus MIN (MAX). Algoritmus pracuje na principu hledání minimální (maximální) hodnoty pixelů v okolí zjišťovaného pixelu. Právě „velikost“ tohoto okolí hraje důležitou roli ve výsledné hodnotě barvy zjišťovaného pixelu, a proto je možné tuto velikost volit pomocí uživatelského rozhraní. Po nalezení minimální (maximální) hodnoty se původní pixel přepíše právě touto nalezenou hodnotou. V následujících kapitolách budeme předpokládat, že potřebujeme zesílit černou linku (čáru) na bílém pozadí. Budeme proto používat algoritmus MIN. Pozn. Kdybychom aplikovali algoritmus MAX, došlo by ke ztenčení černé linky (čáry) na bílém pozadí.

3.1. Způsoby definování okolí bodu

Při hledání minima (maxima) v okolí daného bodu jsem narazil na několik způsobů definování tohoto okolí. Zde je jejich popis:

3.1.1. „Přirozený“ způsob se čtvercovou maticí okolí

V tomto případě je okolí bodu definováno pomocí bodu a velikostí okolí. Pro zjednodušení si toto okolí můžeme představit jako čtvercovou matici o rozměru $2 * velikost + 1$, v jejímž středu je právě zjišťovaný pixel. Pro lepší představu uvedu příklad, kdy velikost zesílení bude 2 pixely. Okolí, ve kterém budeme hledat minimální hodnotu světla, bude představovat matice o rozměrech 5x5 (obr. 3.1.1):

100	120	130	150	200
154	158	168	125	128
125	175	X	158	125
125	180	250	240	126
157	50	60	48	15

Obr. 3.1.1 Ukázka matice okolí

Pokud by jednotlivé buňky matice na obr. 3.1.1 představovaly stupně šedi pixelů, pak by se zjišťovaný pixel X při aplikaci MIN algoritmu obarvil na „nejnižší“ hodnotu šedi, tj. nabyl by hodnotu 15 (od pixelu se souřadnicemi 5;5).

3.1.2. „Přirozený“ způsob s kružnicovým výběrem okolí

Tato definice okolí je podobná předchozí definici okolí. Tentokrát ovšem velikost nebude představovat čtvercovou matici, ale poloměr kružnice. Ve středu této kružnice bude opět ležet zjišťovaný bod. Pro názornost uvedu obrázek, kde se poloměr bude rovnat 2 (obr. 3.1.2):

100				200
	120	130	150	
154	158	168	125	128
125	175	X	158	125
125	180	250	240	126
	50		48	
157		60		15

Obr. 3.1.2 Ukázka okolí, definovaného kružnicí

Jednotlivá čísla v buňkách v obr. 3.1.2 opět představují stupně šedi jednotlivých pixelů v okolí zjišťovaného pixelu. Matice na obr. 3.1.2 obsahuje stejná čísla jako matice na obr. 3.1.1. Při aplikaci MIN algoritmu by ovšem zjišťovaný pixel nabył hodnotu 50, protože hodnota 15 leží „více“ mimo kružnici a není na ní proto brán zřetel.

3.1.3. Dopředné zjišťování okolí

Okolí bodu je opět definováno souřadnicemi zjišťovaného bodu a velikostí okolí. Okolí tentokrát bude představovat obdélníková matice o rozměrech 2 * velikost + 1 a velikost + 1. Zjišťovaný bod bude tentokrát ležet v této matici vlevo uprostřed. Ukázka bude opět pro velikost 2 (obr. 3.1.3):

100	120	130
154	158	168
X	175	158
125	180	250
157	50	60

Obr. 3.1.3 Ukázka dopředného okolí

Při aplikaci MIN algoritmu bude mít zjišťovaný pixel X hodnotu 60.

3.1.4. Zpětné zjišťování okolí

Jedná se o opačný způsob definice okolí oproti dopřednému způsobu zjišťování okolí.

3.1.5. „Dolní“ způsob definování okolí

Analogicky vůči dopřednému a zpětnému okolí lze definovat i dolní okolí bodu. Okolí opět představuje obdélníková matice o rozměrech velikost + 1 a 2 * velikost + 1. Zjišťovaný bod bude ležet uprostřed první řádky matice. Ukázka pro velikost 2 (obr. 3.1.4):

100	120	X	150	200
154	158	168	125	128
125	175	130	158	125

Obr. 3.1.4 Ukázka dolního okolí

Nová hodnota pixelu X po aplikaci MIN algoritmu bude 100.

3.1.6. „Horní“ způsob definování okolí

Jedná se o reverzní způsob vůči dolnímu způsobu definice okolí.

3.2. Zhodnocení jednotlivých způsobů definování okolí

Pro výsledný obrázek je důležité, aby všechny čáry byly zesíleny stejně. Tento efekt nejlépe vytváří algoritmus, který hledá v okolí definovaném pomocí *kružnicového výběru*. Implementace tohoto algoritmu je ovšem nejednoznačná. Nejednoznačnost spočívá v hraničních pixelech. U všech pixelů ležících v hraniční oblasti bychom museli rozhodnout, zda ještě do okolí patří nebo nikoliv. Navíc není možné tento algoritmus jakkoliv urychlit.

Jako alternativa se tedy nabízí hledání v *přirozeném okolí* bodu. Toto okolí je jednoznačně určeno, není zde žádná nejednoznačnost. Algoritmus hledání lze navíc urychlit (viz níže). Tento algoritmus byl implementován ve výsledném pluginu.

Implementování algoritmů využívajících ostatních způsobů definic okolí (*dopředné, zpětné, horní, dolní*) by se nabízelo zejména kvůli menšímu počtu bodů v okolí a tedy i zrychlení zpracování celého obrázku. Tyto algoritmy ovšem nezesílí všechny čáry v obrázku stejně. Vodorovné a svislé čáry zesílí více, než čáry jdoucí „šikmo“. Těchto definic okolí je ale využito při zpracování krajních pixelů v obrázku.

Pseudokód algoritmu zesílení o celé body využívající algoritmu min:

```
uint8 newColor;           //pro uchování barvy, která se uloží do nového obrázku
extern int miraZesileni;  //míra zesílení o celé body
extern uint8[y_max][x_max] obrazek; //původní obrázek
uint8[y_max][x_max] newObrazek; //nový obrázek
//přechod obrázkem
for y:= 0 to y_max do
```

```

for x:= 0 to xmax do
{
    newColor := 255; //maximální hodnota odstínu šedi
    // průchod okolím
    for pomY:= y - miraZesileni to y + miraZesileni do
        for pomX:= x - miraZesileni to x + miraZesileni do
            {
                //hodnota pixelu v okolí je menší než nová barva
                if (obrazek[pomY][pomX] < newColor)
                    newColor = picture[pomY][pomX];
            }
        newObrazek[y][x] := newColor; //uložení nové hodnoty do nového obrázku
    }
}

```

3.3. Urychlení algoritmu zesílení o celé body

V předchozí kapitole jsem vybral algoritmus pro zesílení čar o celé body. Tento algoritmus při hledání výsledné barvy pixelu hledá minimum ve čtvercové matici, v jejímž středu je zjišťovaný bod. Pokud bychom chtěli zpracovávat okolí každého bodu zvlášť, pak bychom při velikosti obrázku 800x600 a velikosti okolí (míře zesílení) např. 3 dostali 23 milionu porovnání, která bychom museli provést.

Pro urychlení výpočtu minima z okolí využijeme skutečnost, že okolí bodu je matice pixelů. Jednoduše můžeme vypočítat minimum z každého sloupce matice a následně minimum z těchto „sloupcových“ minim. Pokud bychom ale počítali tato „sloupcová“ minima u každého pixelu zvlášť, algoritmus bude stále stejně „pomalý“.

Vyšší efektivity algoritmu dosáhneme, pokud si tato sloupcová minima budeme „pamatovat“ a využijeme jich při výpočtu minima z okolí u následujícího pixelu. Okolí sousedních pixelů se totiž liší pouze v „jednom“ sloupci. Pro nový pixel tedy stačí vypočítat minimum z jednoho sloupce okolí, ostatní sloupce jsou již vypočteny od předchozího pixelu. Pro obrázek o rozměrech 800x600 a velikosti okolí 3 bychom dostali 3,4 milionu porovnání. Počet operací se zmenšil přibližně 7x. Celý proces zrychlení demonstruje obrázek 3.3.1.

100	120	130	150	200	100
154	158	168	125	128	154
125	175	X	X+1	125	125
125	180	250	240	126	125
157	50	60	48	15	157

Obr. 3.3.1 Ukázka výpočtu okolí

Obrázek ukazuje výpočet minima okolí pro body X a $X+1$ při velikosti okolí 2. Pro bod X je třeba spočítat minima z červeného a fialových sloupců a následně určit „sloupcové“ minimum. Pro bod $X+1$ následně stačí jen dopočítat minimum z modrého sloupce a „sloupcové“ minimum počítat z již spočtených fialových sloupců a nově spočteného modrého sloupce.

Ukázky zesílení čar v obrázku. Na obr. 3.4.1 je vidět originální perokresba.



Obr. 3.4.1 Originální obrázek

Míra zesílení o 1 pixel je vidět na obr. 3.4.2



Obr. 3.4.2 Míra zesílení o 1 pixel

Míra zesílení o 4 pixely (obr. 3.4.3):



Obr. 3.4.3 Míra zesílení 4 pixely

Pseudokód upraveného algoritmu zesílení o celé body využívající algoritmus min:

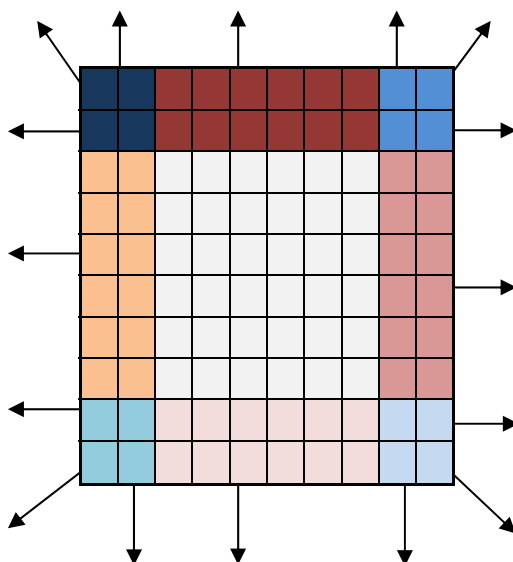
```
extern int miraZesileni; //míra zesílení o celé body
extern uint8[y_max][x_max] obrazek; //původní obrázek
uint8[uint8[y_max][x_max] newObrazek; //nový obrázek
uint8[miraZesileni * 2] sloupcovaMinima; //pole pro uchování sloupcových minim
uint index = 0; //index v tomto poli
for y:= 0 to y_max do //průchod obrázkem
{
    //výpočet prvotních hodnot do pole sloupcových minim
    for index := 0 to (sloupcovaMinima.Length / 2) do
    {
        sloupcovaMinima[index] :=
            minimumZeSloupce(obrazek[y-miraZesileni .. y+miraZesileni][index]);
    }
    for x:= 0 to x_max do
    {
        newObrazek[y][x] := minZpole(sloupcovaMinima); //uložení nových hodnot
        sloupcovaMinima[++index] := minimumZeSloupce(
            obrazek[y-miraZesileni .. y+miraZesileni][x+miraZesileni]);
        if (index == sloupcovaMinima.Length - 1)
            index := 0;
    }
}
}
```

4. Zesilování čar o desetiny (setiny) bodu

V některých případech se může stát, že zesílení čar o celé body nepřichází v úvahu. Důvodem může být příliš velké zesílení, které později ruší celkový dojem z perokresby. Je proto vhodné linky zesílit o zlomky bodu, ne o celý bod. Pokud bychom chtěli linky zesílit např. o desetinu bodu, museli bychom celý obrázek 10x zvětšit, provést zesílení o jeden bod, následně celý obrázek zmenšit na původní velikost. Tento postup je ale časově a paměťově náročný. Navíc, kdybychom chtěli linky v obrázku opětovně zvětšit o další desetinu bodu, byli bychom nuceni celý postup opakovat.


Předpokládejme, že toto zesílení chceme provést v jednom kroku a nejlépe bez zvětšování obrázku. Tohoto „přání“ lze dosáhnout, pokud si pixel nebudeme představovat jako jednotlivý bod, ale jako čtvercovou matici. V případě, že bychom chtěli zesílovat o desetiny bodu, pak by tato matice měla rozměr 10x10. V případě zesílení o setiny bodu, by tato matice měla rozměry 100x100. Pro jednoduchost budu dále uvažovat se zesílením o desetiny bodu.

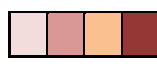
Při počítání minim je důležité si představit, že okolní pixely jsou reprezentovány také maticí o stejné velikosti, jako má matice zjišťovaného pixelu. Míra zesílení se v tomto případě stává celým číslem (původní desetinné se vynásobí deseti). Nyní lze aplikovat algoritmus min pro každou buňku matice zjišťovaného pixelu. Výsledná hodnota pixelu vznikne aritmetickým průměrem všech hodnot matice pixelu. Algoritmus výpočtu bude nastíněn v následujícím obrázku (obr. 4.1) pro míru zesílení 0,2.




Obr. 4.1 Míra zesílení 0,2

Na obr. 4.1 je vidět, které buňky používají pro výpočet minim i okolní pixely. Šipky reprezentují odkaz na okolní pixely.

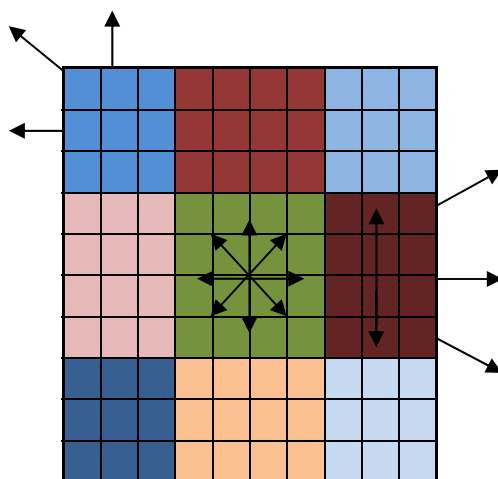
 Bílé buňky uprostřed si zachovávají původní hodnotu

 Odstíny červené představují buňky, které počítají minimum ze své současné hodnoty a ze svého souseda

 Odstíny modré představují buňky, které počítají minimum „ze všech sousedících pixelů“.


Matice pro míru zesílení 0,1 – 0,4 jsou si podobné. „Podobností“ se rozumí, že všechny tyto matice obsahují pouze „tyto barvy“. V matici pro zesílení o 0,5 bodu se vyskytují pouze buňky o odstínech modré.


Matice pro zesílení o více než 5 desetín bodu obsahuje „jiné“ buňky. Je to dáno tím, že pro všechny buňky je nutné počítat minimum z okolí. Ukázka pro míru zesílení 0,7 je vidět na obr. 4.2.




Obr. 4.2 Míra zesílení 0,7

V obr. 4.2 je opět vidět, ze kterých pixelů je spočteno minimum.

 Zeleně obarvené buňky počítají svou hodnotu z celého okolí pixelu.

 Odstíny červené představují buňky, které minimum počítají ze všech směrů kromě směru jdoucího přes střed.

 Odstíny modré představují buňky, které počítají minimum jen ze svých aktuálních sousedů.

Pozn. Opět si jsou matice pro míru zesílení 0,6 – 0,9 podobné, tj. obsahují pouze takto barevné buňky.

Pseudokód algoritmu zesílení o desetiny bodu využívající min algoritmu:

```
extern int miraZesileni * 10;           //původní hodnota byla z intervalu <0,1; 0,9>
extern uint8[y_max][x_max] obrazek;    //původní obrázek
uint8[y_max][x_max] newObrazek;       //nový obrázek
uint8[10][10] maticePixelu;           //matice pixelu
//průchod obrázkem
for y:= 0 to y_max do
  for x:= 0 to x_max do
    {
      //převedením do matic se rozumí vytvoření matic o rozměrech 10x10
      //a vyplnění těchto matic hodnotami příslušného pixelu
      prevedPixelyDoMaticPixelu(obrazek[y-1..y+1][x-1..x+1]);
      for i := 0 to 10 do
        for j := 0 to 10 do
          {
            //okolní pixely jsou zde prezentovány maticemi 10x10
            maticePixelu[i][j] := spocitiMinZokolnichMaticPixelu(
              i-miraZesileni..i+miraZesileni, j-miraZesileni..j+miraZesileni);
          }
      newObrazek[y][x] := soucetHodnot(pixelMatice) / (10 * 10);
    }
  }
```

4.1. Spočítání nové hodnoty pixelu bez nutnosti vyplnění celé matice

Při bližším pohledu na matice je patrné, že obsahují „nějaké“ submatice, které mají po vyplnění celé matice stejnou hodnotu (na obr. 4.1 a 4.2 jsou zobrazeny stejnou barvou). Pokud bychom určili rozměry těchto submatic, stačilo by následně spočítat pouze hodnotu jednoho pixelu v této submatici a vynásobit jej počtem buněk. Určení výsledné hodnoty pixelu již z takto vypočtených hodnot je jednoduché.

Výpočet rozměrů jednotlivých submatic je třeba rozdělit podle míry zesílení. Pro míru z intervalu <0,1; 0,5> uvádí rozměry submatic rovnice 4.1.1, pro míru z intervalu <0,6; 0,9> uvádí rozměry submatic rovnice 4.1.2. Pozn. Barvy jednotlivých submatic jsou převzaty z obr. 4.1 a 4.2.

Rozměry bílé submatice:

$$X = 10 - 2 * (míra * 10), \quad Y = 10 - 2 * (míra * 10)$$

Rozměry „červených“ submatic:

$$X = 10 - 2 * (míra * 10), \quad Y = míra * 10$$

Rozměry „modrých“ submatic:

$$X = (míra * 10), \quad Y = (míra * 10)$$

Rovnice. 4.1.1 Výpočty rozměrů submatic pro míru $\in < 0,1; 0,5 >$

Rozměry zelené submatice:

$$X = 2 * (míra * 10) - 10, \quad Y = 2 * (míra * 10) - 10$$

Rozměry „červených“ submatic:

$$X = 10 - (míra * 10), \quad Y = 2 * (míra * 10) - 10$$

Rozměry „modrých“ submatic:

$$X = 10 - (míra * 10), \quad Y = 10 - (míra * 10)$$

Rovnice. 4.1.2 Výpočty rozměrů submatic pro míru $\in < 0,6; 0,9 >$

Pozn. Pro zesílení o setiny bodů stačí ve výše uvedených vzorcích (obr. 4.1.1 a 4.1.2) zaměnit číslo 10 za 100. Interval v obr. 4.1.1 by poté byl $<0,01; 0,50>$, interval v obr. 4.1.1 by byl $<0,51; 0,99>$.

4.2. Zvýšení efektivity algoritmu zesilování o desetiny (setiny) bodu

Výše uvedený algoritmus vyhledává minima pro každou submatici vždy z předem určených sousedních pixelů. Stejně jako v případě zesilování o celé body, i zde můžeme vypočítat „sloupcová minima“ a následně pro každou submatici určit minimum z příslušných sloupcových minim. Zde ovšem nemůžeme počítat minima z celého sloupce, jako tomu bylo u zesílení o celé body. Sloupce je potřeba rozdělit na díly, které budou uchovávat minima z příslušných sousedních pixelů.

Tyto díly je potřeba vhodně zvolit, aby byly použitelné pro výpočet minima z více než jedné submatice matice pixelu. Nejvhodnější rozdělení je vidět na obr. 4.2.1.

Původní matice (zjišťovaný pixel je označen písmenem X):

a	d	f
b	X	g
c	e	h

Rozdělení prvního sloupce:

a
b

b
c

Rozdělení druhého sloupce:

d
X

X
e

Rozdělení třetího sloupce:

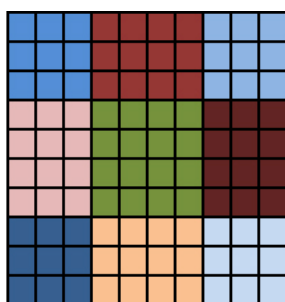
f
g

g
h

Obr. 4.2.1 Rozdělení sloupců matice

Po výpočtu minim z takto rozdělených sloupců dostaneme celkem šest hodnot, které téměř pokryjí požadavky na výpočty hodnot v jednotlivých submaticích. Při výpočtu jednotlivých submatic se vždy vypočte minimum z příslušných dílů rozdělení matice. Výjimku tvoří výpočet minim z pravého a levého pixelu, tj. minimum z b-X a minimum z X-g (viz obr. 4.2.1). Tato minima se ovšem použijí jen u výpočtu jedné submatice, proto není nutné jejich hodnotu uchovávat. Příklad výpočtu viz obr. 4.2.2.

Matice pro míru našíření 0,7:



Pixely ze submatice

 se vypočítají z dílů

a
b

d
X

Pixely ze submatice

 se vypočítají z dílů:

a
b

d
X

f
g

Obr. 4.2.2 Ukázka výpočtu pixelů ze submatic

Pozn. Pixely z ostatních submatic se vypočítají analogicky k výpočtům z obr. 4.2.2. K určení dílů, ze kterých se mají pixely počítat, poslouží obr. 4.2

Zrychlení algoritmu je možné díky opětovnému použití již vypočtených hodnot. Po výpočtu nové hodnoty pixelu se přesuneme na vedlejší pixel. Pro zjištění hodnot submatic

tohoto pixelu můžeme použít již vypočtených hodnot z předchozího pixelu. Zbývá nám pouze vypočítat dvě nové hodnoty pro díly, které jsou pro tento pixel nové.

Ukázky zesílení obrázku o desetiny bodu. První obrázek je originál (obr. 4.3.1)



Obr. 4.3.1 Originál

Míra zesílení o 0,5 pixelu (obr. 4.3.2)



Obr. 4.3.2 Míra zesílení 0,5 pixelu

Míra zesílení o 0,9 pixelu (obr. 4.3.3)



Obr. 4.3.3 Míra zesílení 0,9 pixelu

Pseudokód efektivního algoritmu zesílení o desetiny bodu využívající min algoritmu:

```

extern int miraZesileni * 10;           //původní hodnota byla z intervalu <0,1; 0,9>
extern uint8[y_max][x_max] obrazek;    //původní obrázek
uint8[y_max][x_max] newObrazek;       //nový obrázek
uint8[2] vlevo;                        //uchování minim „z levých pixelů“
uint8[2] pozice;                       //uchování minim z horního a dolního pixelu
uint8[2] vpravo;                       //uchování minim „z levých pixelů“
uint zCeleho, rohy, meziRohy;         //“konstanty“ pro velikost submatic
uint newColor;                         //pro uchování nove barvy

//pro miruZesileni ∈ < 0,1; 0,5 >
zCeleho := (10-(2*miraZesileni))^2;
rohy := mira^2;
meziRohy:= (10-
(2*miraZesileni))*miraZesileni;

//průchod obrázkem
for y:= 0 to y_max do
{
    vlevo[0]:= min(obrazek[y-1..y][x-1]);
    vlevo[1]:= min(obrazek[y..y+1][x-1]);
    pozice[0]:= min(obrazek[y-1..y][x]);
    pozice[1]:= min(obrazek[y..y+1][x]);
    vpravo [0]:= min(obrazek[y-1..y][x+1]);
    vpravo [1]:= min(obrazek[y..y+1][x+1]);
}
//pro miruZesileni ∈ < 0,6; 0,9 >
zCeleho :=(2*mira - 10)^2;
rohy := (10 - mira)^2;
meziRohy:=(10 - miraZesileni) *
(2*miraZesileni - 10);

```

```

for x:= 0 to xmax do
{
    newColor := rohy * (vlevo[0] * pozice[0] + vlevo[1] * pozice[1] + vpravo[0] *
        pozice[0] + vpravo[1] * pozice[1]);

//pro miruZesileni ∈ < 0,1; 0,5 >
newColor += meziRohy * (min(obrazek[y][x-
1..x] + min(obrazek[y][x..x+1]) +
min(obrazek[y-1..y][x] +
min(obrazek[y..y+1][x]);
newColor += zCeleho * obrazek[y][x]);

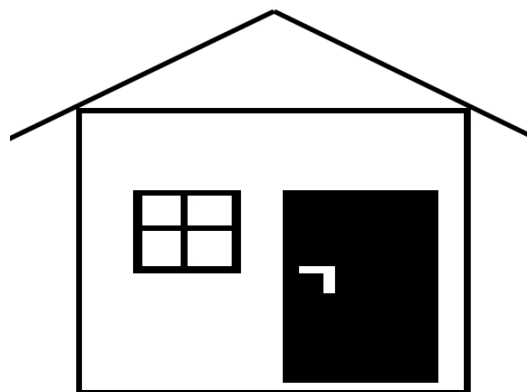
//pro miruZesileni ∈ < 0,6; 0,9 >
newColor += meziRohy * (min(vlevo[0],
pozice[0], vpravo[0]) * min(vlevo[1],
pozice[1], vpravo[1]) + min(vlevo[0..1],
pozice[0..1]) + min(pozice[0..1],
vpravo[0..1]));
newColor += zCeleho * min(vlevo, pozice,
vpravo);

    newObrazek[y][x] := newColor / (10*10);
    vlevo := pozice; pozice := vpravo;
    vpravo[0] := min(obrazek[y-1..y][x+2];
    vpravo[1] := min(obrazek[y..y+1][x+2];
}
}

```

5. Detekce pozadí (pozitivní x negativní zesilování čar)

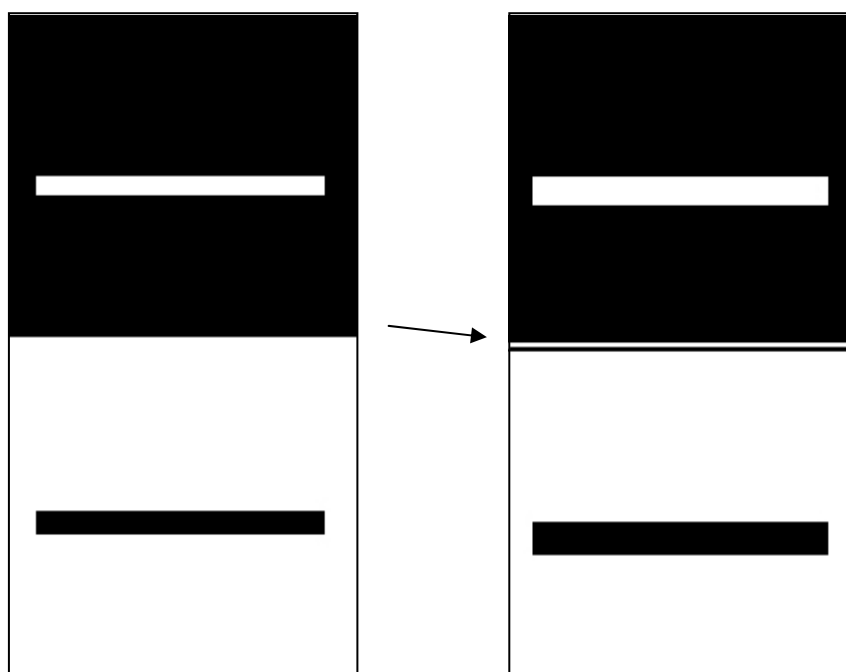
Dosud jsme se soustředili na zesílení černých čar na bílém pozadí. Tento způsob zesilování se jmenuje pozitivní zesilování čar. Některé obrázky ovšem mohou obsahovat bílé čáry na černém pozadí. Pro zesílení těchto čar je potřeba použít algoritmus MAX, který v okolí jednotlivých pixelů nehledá minimální hodnotu, ale naopak maximální hodnotu. Takto popsaný algoritmus se jmenuje negativní zesilování čar. Zvláštní případ nastane, bude-li obrázek obsahovat jak černé čáry na bílém pozadí, tak i bílé čáry na černém pozadí. Ukázka takového obrázku - viz obr. 5.1.



Obr. 5.1 Ukázka obrázku s bílými čarami na černém pozadí a černými čarami na bílém pozadí

V obrázku je tedy nutné rozhodnout, který algoritmus se v jakých částech obrázku použije. Pro každý zjišťovaný pixel je potřeba udělat statistiku, kolik černých a kolik bílých pixelů se v jeho okolí vyskytuje a následně určit algoritmus, který provede zesílení čar (spočte novou hodnotu zjišťovaného pixelu). V případě, že počet bílých pixelů přibližně odpovídá počtu černých pixelů, algoritmus pouze „zkopíruje“ hodnotu původního pixelu, tj. neprovede zesílení.

Okolím bodu se opět rozumí „přirozené“ okolí bodu. Velikost tohoto okolí je možné zadat pomocí grafického uživatelského prostředí. Výpočet sumy černých a bílých pixelů se provádí pomocí podobného algoritmu jako zesílení o celé body. Opět se postupně prochází všechny pixely v okolí. Lze zde použít i stejného urychlení jako u algoritmu zesilování o celé body. Algoritmus nejprve spočte počet černých a počet bílých pixelů v jednotlivých sloupcích okolí, následně sečte počet bílých a počet černých pixelů. Vyhodnocením statistických údajů se rozumí porovnání těchto dvou hodnot. Uživatel si sám může pomocí uživatelského rozhraní zvolit, při jaké procentuální „převaze“ pixelů proběhne algoritmus zesílení. Původně bylo „napevno“ určeno, že tato hranice bude 50%, algoritmus se však na rozhraní negativního / pozitivního zesilování stával nestabilním (viz obr. 5.2). Pomocí uživatelského rozhraní je tedy možné tuto hranici zadat, a to v rozmezí 50%-100%. Dle vlastních měření je algoritmus stabilní pro hodnoty z intervalu 60% – 75%. Při hodnotách vyšších než 75% často k zesílení čar vůbec nedojde, což je opět nepříznivý stav.



Obr. 5.2 Nestabilita algoritmu na rozhraní při zesilování o 2 body

Ukázky aplikace algoritmu zesilování s a bez detekce pozadí na obrázcích, převaha pixelů byla zvolena na 66% (obr. 5.3 – 5.4)



Obr. 5.3 Originální obrázek



Obr. 5.4 Zesílení o 2,5 bodu bez detekce pozadí



Obr. 5.5 Zesílení o 2,5 bodu s detekcí pozadí

Pseudokód algoritmu detekce pozadí:

```
extern uint8[ymax][xmax] obrazek;           //obrázek
extern uint velikostPozadi;
extern uint procentaulniPrevaha;
uint[velikostPozadi * 2] pocetBile = {0}; //pole pro uchování počtu bílých pixelů
uint[velikostPozadi * 2] pocetCerne = {0}; //pole pro uchování počtu černých pixelů
uint index = 0;                               //index v těchto polích
//průchod obrázkem
for y:= 0 to ymax do
{
    for index := 0 to pocetBile.Length / 2 do
    {
        pocetBile [index] := spoctiPocetBilych(
            obrazek[y-velikostPozadi..y+velikostPozadi][index]);
        pocetCerne [index] := spoctiPocetCernych(
            obrazek[y-velikostPozadi..y+velikostPozadi][index]);
    }
    for x:= 0 to xmax do
    {
        int bilych := sumaPole(pocetBile);
        int cernych := sumaPole(pocetCerne);
        if (bilych * procentaulniPrevaha / 100 > cernych)
            pouzijMinAlgoritmus();
        else if (cernych * procentaulniPrevaha / 100 > bilych)
            pouzijMaxAlgoritmus();
        else kopirujHodnotu();
        pocetBile [++index] := spoctiPocetBilych(
            obrazek[y-velikostPozadi..y+velikostPozadi][x+velikostPozadi]);
        pocetCerne [index] := spoctiPocetCernych(
            obrazek[y-velikostPozadi..y+velikostPozadi][ x+velikostPozadi]);
        if (index == pocetBile.Length - 1)
            index := 0;
    }
}
}
```

6. Známé chyby algoritmu

Na začátku této kapitoly bych rád uvedl slovní spojení „nic není dokonalé“. I pro výše popsané algoritmy je možné vytipovat perokresby, na nichž algoritmus nebude pracovat

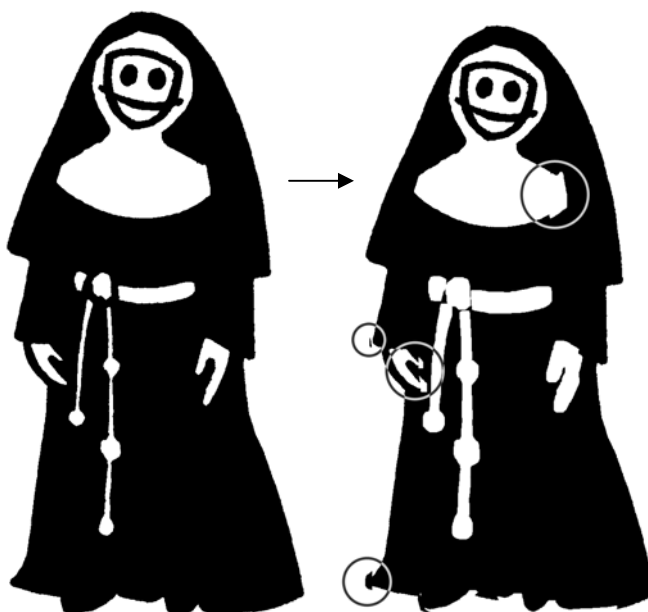
správně. Konkrétně se chyby mohou vyskytnout při použití detekce pozadí. Obrázek může obsahovat různé přechody mezi pozadími, a proto by každá jeho část potřebovala provést detekci pozadí s různými nastaveními pro velikost detekčního okolí pixelu a prahu převahy počtu pixelů pro aplikaci algoritmu zesílení. Příklad takového obrázku je vidět na obr. 6.1.



Obr. 6.1. Obrázek „Duchovní“

Na obrázku 6.1 jsou vyobrazeny tři objekty – kněz, kostel a jeptiška. Při pokusu o zesílení o 6 pixelů a při detekci pozadí vykazuje algoritmus největší nestabilitu při zesilování linek na jeptišce, proto se v následujícím textu zaměřím právě na ni.

Následující obrázek (obr. 6.2) ukazuje pokus o zesílení o 6 pixelů, detekce pozadí na okolí o velikosti 140 pixelů a převaze pixelů 75%.



Obr. 6.2 Zesílení čar o 6 pixelů při detekci pozadí

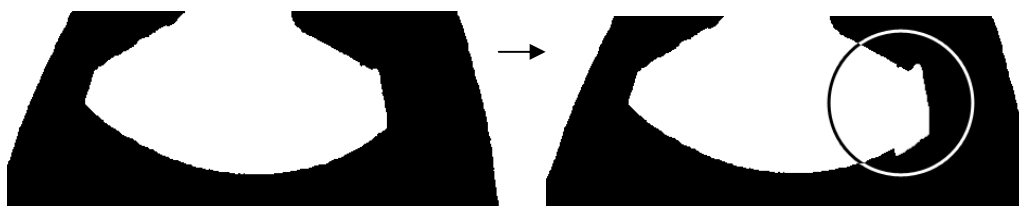
Nyní budou popsány chyby, které se vyskytly při výše uvedeném zesílení včetně nástinu jejich řešení. Hlavním cílem nastavení bylo provést zesílení pásku a rukou (bílých částí na černém pozadí).

- Nedošlo k zesílení linek v obličejí

Tato chyba nastala kvůli velkému nastavení procentuálního prahu při detekci pozadí. Algoritmus při vyhodnocení statistik počítá i s černými pixely v pokrývce hlavy. Nikdy tedy nedojde k 75% převaze černých nebo bílých pixelů, pixely se vždy pouze zkopírují. Možné řešení tohoto problému je pouze v „manuálním“ zesílení linek v obličejí, tzn. označit linky v obličejí a následně a provést jejich zesílení.

- Došlo k zesílení okraje pravé části bílé plochy pod hlavou jeptišky

Tento artefakt je ve zvětšenině vidět na obr. 6.3. K této chybě došlo kvůli nízkému procentuálnímu prahu počtu pixelů při detekci pozadí. V této oblasti dochází k velké převaze černých pixelů. Algoritmus detekce pozadí vyhodnotí tuto situaci a výsledkem je zesílení bílých čar na černém pozadí s následným vznikem tohoto artefaktu. Možné řešení je v nastavení většího procentuálního prahu pro detekci pozadí (minimálně 80%).



Obr. 6.3 Artefakt v levé části bílé plochy

- Artefakty na rozhraní mezi černým a bílým pozadím

Tyto artefakty jsou ve zvětšenině vidět na obrázku 6.4. K těmto chybám došlo opět kvůli nízkému procentuálnímu prahu počtu pixelů při detekci pozadí. V těchto oblastech přechodu dochází k velké převaze bílých pixelů. Algoritmus detekce pozadí vyhodnotí tuto situaci, dojde k zesílení černých čar na bílém pozadí a následně vzniknou tyto artefakty. Možné řešení je opět v nastavení většího procentuálního prahu pro detekci pozadí (minimálně 80%).



Obr. 6.4 Artefakty na rozhraní mezi pozadími

- Špatné zesílení pravé ruky jeptišky

Tento artefakt vznikl kvůli malé vzdálenosti ruky od přechodu mezi bílým a černým pozadím. Skoro celá ruka leží při aplikaci algoritmu detekce pozadí v oblasti, kde se pixely „pouze kopírují“. Na tomto artefaktu je přesně vidět, kdy došlo k 75% převaze černých pixelů a spustil se tedy algoritmus zesílení bílých linek na černém pozadí. Možné řešení tohoto problému vidím jedině v manuálním zesílení kresby pravé ruky jeptišky, tzn. vybrat linky v pravé ruce a provést zesílení. Artefakt je ve zvětšenině vidět na obr. 6.5.



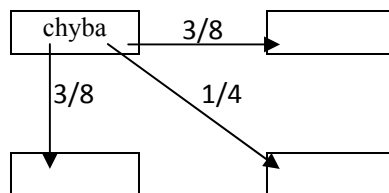
Obr. 6.5 Artefakt na pravé ruce

Pozn. Při různých nastaveních detekce se snažíme o odstranění artefaktů, aby výsledný obrázek byl „co nejlepší“. Při pokusech na výše uvedeném obrázku se mi ovšem nepodařilo najít takové nastavení, které by fungovalo na celém obrázku. Pro nejlepší výsledek je nutné provést detekci pozadí „manuálně“.

7. Příprava na tisk

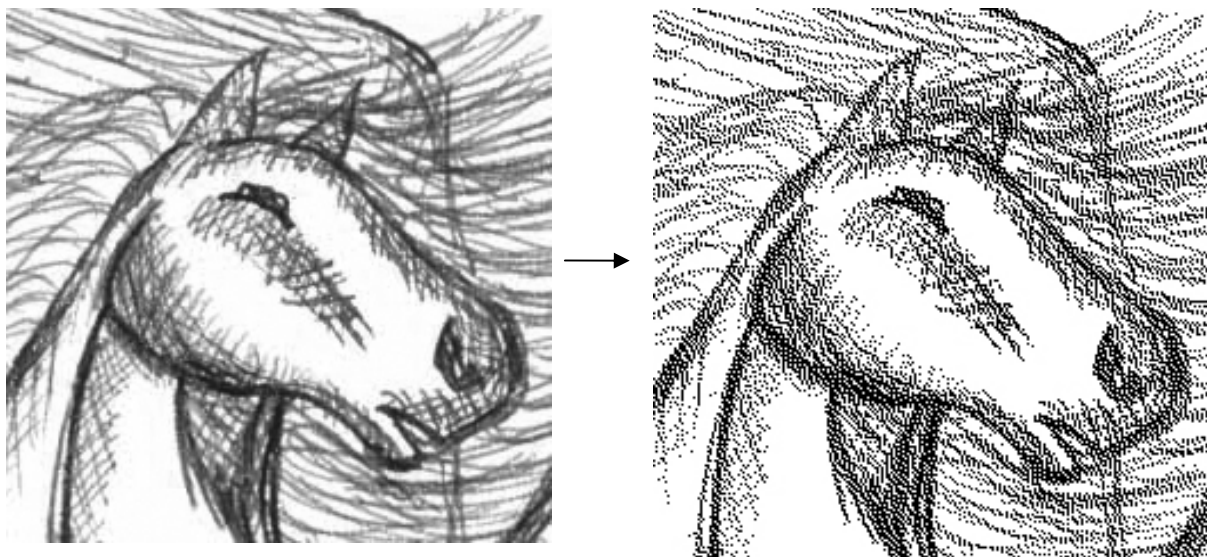
Tento plugin má sloužit především pro úpravu perokreseb pro tisk. Nejprve je potřeba čáry v perokresbě zesílit. Algoritmus zesílení ovšem produkuje i jednotlivé odstíny šedi, které není možné přímo tisknout pomocí černobílých tiskáren. Plugin tedy volitelně umožňuje úpravu šedotónového obrázku do černobílého, tzv. dithering. Ve svém pluginu jsem zvolil metodu Floyd-Steinbergova ditheringu. Pozn. Photoshop také nabízí možnost převedení na černobílý obrázek pomocí prahové funkce. Záleží na uživateli, kterou z nabízených možností si pro výslednou úpravu perokresby zvolí.

Floyd-Steinbergův algoritmus pracuje na základě prahu a distribuce chyby. Předpokládáme šedotónový obrázek. Každý pixel tohoto obrázku je zobrazen jedním bytem. Může tedy nabývat hodnot z intervalu $\langle 0; 255 \rangle$. Hodnotou prahu se většinou rozumí střední hodnota tohoto intervalu, v tomto případě 127. Hodnoty menší než prah se zobrazí na 0, hodnoty větší nebo rovny prahu se zobrazí na 255. Hodnotu prahu je možné volit pomocí grafického uživatelského prostředí. Chyba, která vznikne rozdílem nové hodnoty pixelu a původní hodnoty, se distribuuje do okolních pixelů. Směr a velikost distribuované chyby zobrazuje následující obrázek (obr. 6.1).



Obr. 6.1 Distribuce chyby ve Floyd-Steinbergově algoritmu

Následující obrázek (obr. 6.2) ukazuje aplikaci Floyd-Steinbergova algoritmu na šedotónový obraz.



Obr. 6.2 Aplikace Floyd-Steinbergova algoritmu na šedotónový obrázek

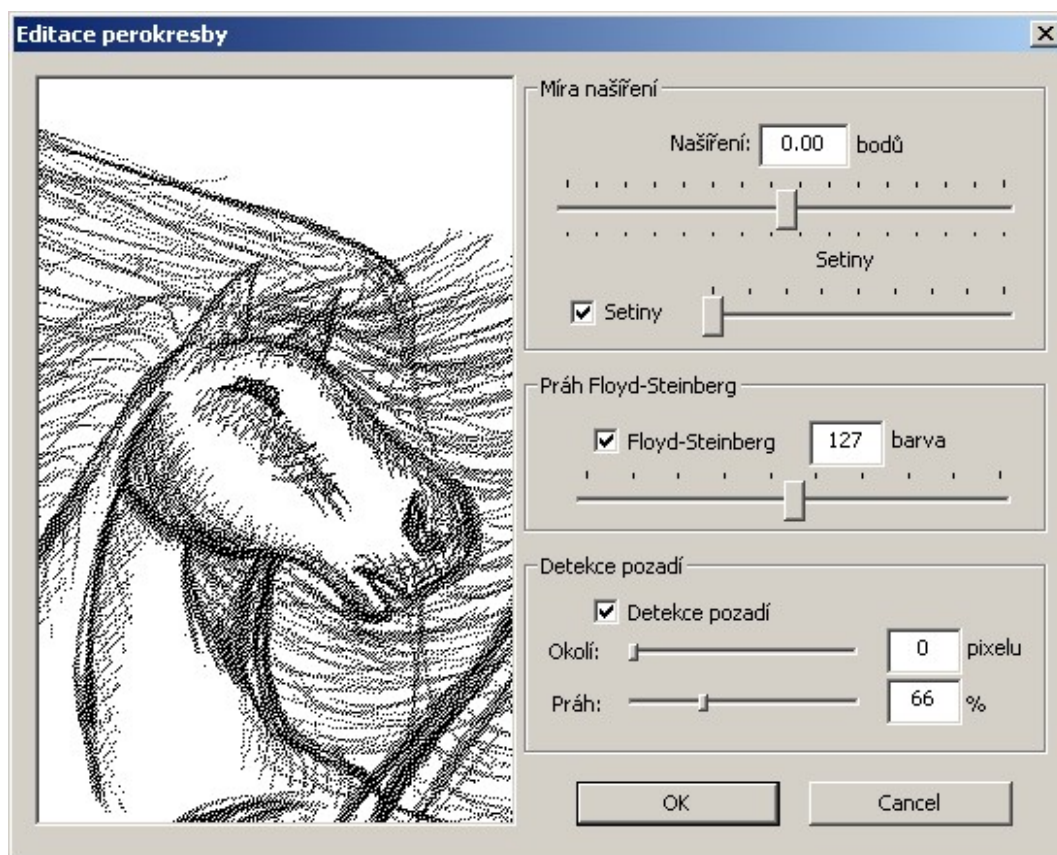
8. Plugin pro Adobe Photoshop a tvorba dialogu

8.1. Plugin

Tato kapitola není zaměřena na podrobný popis a základní nastavení pro vytvoření vlastního pluginu, jak by se dalo očekávat. Informace lze získat v publikaci pana Ing. Jiřího Skály „Tvorba zásuvných modulů pro Adobe Photoshop“, ve které je v kapitolách 2 až 8 popsáno vše podstatné pro vytvoření pluginu do aplikace Adobe Photoshop pomocí vývojového prostředí Microsoft Visuál C++ .

Považuji za nutné se ale zmínit o tom, že mnou vytvořený plugin patří do kategorie zásuvných modulů typu Filtr a proto jej při použití naleznete v menu Filtr\Perokresba. Po spuštění se zobrazí dialog s uživatelským rozhraním (viz obr. 7.1), který slouží k nastavení parametrů pro editaci perokresby. Trojice panelů - Míra našíření, Práh Floyd-Steinberg a Detekce pozadí - jsou po řadě určeny k nastavení editace perokresby. Na dialogu je možné v

levé části naleznete okno náhledu. Do tohoto okna se zobrazuje část obrazu, která se do něj vejde, po aplikaci filtru. Zároveň je umožněno měnit v něm zobrazovanou část.



Obr. 7.1 Dialogové okno pluginu

Dále je nutno zdůraznit, kde naleznete důležité adresáře s hlavičkovými soubory pro tvorbu pluginu. Pokud zachováte základní nastavení při instalaci Adobe Photoshop SDK 6.0, umístí se do adresáře: „*c:\Program Files\Adobe\Adobe Photoshop 6.0 SDK*“, ze kterého pro nás budou důležité soubory v adresářích:

- .. *\Common\Includes*
- .. *\Common\Resources*
- .. *\PhotoshopAPI\General*
- .. *\PhotoshopAPI\Photoshop*
- .. *\PhotoshopAPI\Pica_sp*

8.2. Dialog

Při vytváření uživatelského rozhraní můžeme volit mezi implementací pomocí Adobe Dialog Manageru a Win32Api.

Adobe Dialog Manager je rozhraní pro správu dialogových oken v aplikacích Adobe. Dialogy vytvořené pomocí Adobe Dialog Manageru jsou nezávislé na platformě a mají konzistentní vzhled, tzv. look and feel.

Já jsem se rozhodl použít Win32Api s pomocí šablony, která se ukládá do souboru s koncovkou „.rc“. Výhodu tvorby uživatelského rozhraní pomocí Win32Api vidím v dostupnosti materiálů. Nevýhodu spatřuji právě v následné závislosti na platformě Microsoft Windows. Vytvoření formuláře a veškeré obsluhy událostí od kontrolků na formuláři se nachází v souboru „dialog.cpp“. Získání handlu modulu pro dialog se nachází v souboru „zmenaMain.cpp“. Myslím si, že oba tyto zdrojové soubory jsou v dostatečné míře okomentovány. Jednotlivé informace je možné získat právě z těchto zdrojových souborů.

9. Uživatelská příručka

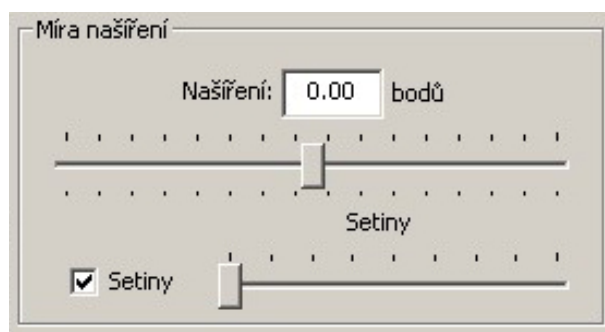
Nejprve je třeba nakopírovat soubor „EditacePerokresby.8bf“ do adresáře Adobe Photoshopu se jménem Plug-Ins\Filters. Užíváte-li českou verzi programu, pak jej zkopírujte do adresáře Zásuvné moduly\Filtry. Plugin po novém spuštění Photoshopu naleznete v menu Filtr\Perokresba>Editace perokresby. Po spuštění pluginu se zobrazí dialogové okno (obr. 7.1), které můžete kdykoli ukončit pomocí tlačítka Cancel, křížku v levém horním rohu nebo klávesy Esc.

V levé polovině dialogu naleznete okno s náhledem obrázku. Po najetí kurzoru do prostoru vymezeného pro náhled se kurzor myši změní do podoby ruky, čímž vás informuje o možnosti pohybovat obrázkem v náhledu. Změnu polohy docílíte přidržením levého tlačítka myši a jejím pohybem. Při pohybu náhledu mění kurzor svou podobu do tvaru čtyřsměrné šipky.

V pravé části dialogu se nachází panel pojmenovaný **Míra našíření** a dále dvě zaškrtnutá políčka pojmenovaná **Floyd-Steinberg** a **Detekce pozadí**.

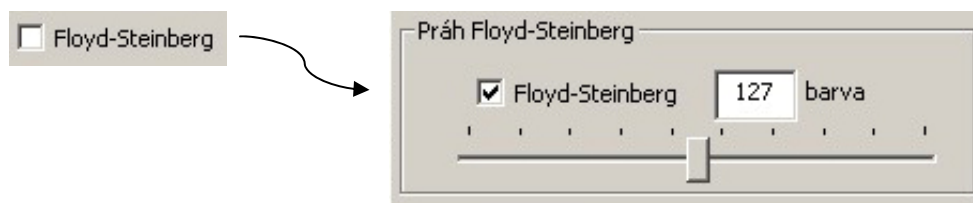
V panelu Míra našíření je možné editovat, „o kolik pixelů“ se mají zesílit nebo zeslabit čáry v obrázku. Panel Míra našíření ukazuje obr. 8.1. „Editovat“ je možné buď taháním příslušného slideru, který je přítomný v panelu, nebo přímým zápisem hodnoty do editboxu. Zadávaná hodnota musí být v intervalu <-15;15>. Pokud není, nastaví se standardní hodnota, nula. Pokud se nacházíte v editboxu pro míru našíření, je možné měnit hodnotu míry i stiskem šipek. Šipka nahoru a dolů přidává, respektive ubírá setinu od aktuální hodnoty, šipka doleva a doprava ubírá, respektive přidává desetinu k aktuální hodnotě míry. Uvažujete-li o nastavení

míry zesílení v setinách pixelu pomocí myši, je potřeba nejprve zaškrtnout tlačítko Setiny, následně se objeví slider, který toto nastavení umožňuje.



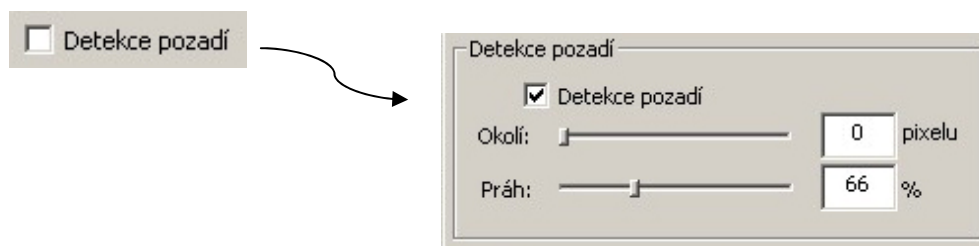
Obr. 8.1 Panel Míra našíření

Chcete-li na výsledný obrázek použít dithering, je možné použít zabudovaný Floyd-Steinbergův algoritmus. Algoritmus se zapíná pomocí zaškrťovacího políčka Floyd-Steinberg. Po jeho zaškrtnutí se objeví panel Práh Floyd-Steinberg (obr. 8.2), ve kterém je možné editovat velikost prahu. Práh je potřeba zadat z intervalu $\langle 100;154 \rangle$. Jeho hodnotu je opět možné zadat přímo do editboxu nebo použít slider. Pokud zadaná hodnota v editboxu bude mimo tento interval, nastaví se jako hodnota bližší z krajních hodnot požadovaného intervalu. I zde, pokud se nacházíte v editboxu, je možné použít šipky. Tentokrát šipky nahoru a doprava přidávají „jeden odstín šedi“, šipky dolů a doleva naopak odstín ubírají.



Obr. 8.2 Zobrazení panelu Práh Floyd-Steinberg

Pokud je v obrázku potřeba provádět pozitivní i negativní zesilování (ztenčování) čar, je nutné na formuláři zaškrtnout políčko Detekce pozadí. Po jeho zaškrtnutí se objeví panel s nastavením pro detekci pozadí (obr. 8.3). Na tomto panelu je možné nastavit velikost okolí, na kterém se bude provádět statistika pixelů, a dále nastavit při jaké procentuální převaze pixelů se algoritmus aplikuje. Nastavení okolí je opět možné pomocí slideru nebo přímým zadáním hodnoty do editboxu. Při přímém zadávání je potřeba zadat hodnotu z intervalu $\langle 0;150 \rangle$. Pokud je zadaná hodnota mimo tento interval, nastaví se bližší z krajních hodnot intervalu. Opět je možné pro zvýšení/snížení hodnoty použít šipky nahoru, doprava, respektive dolů, doleva. Nastavení procentuální převahy pixelů (prahu) je možné pomocí slideru nebo pomocí editboxu. Při zadávání hodnoty do editboxu je potřeba zadat hodnotu z intervalu $\langle 50;100 \rangle$. Pokud zadaná hodnota není z tohoto intervalu, pak se opraví na bližší krajní hodnotu z tohoto intervalu.



Obr. 8.3 Zobrazení panelu Detekce pozadí

10. Závěr

Při zpracování zadané bakalářské práce jsem se nejprve musel seznámit s algoritmem pro zesilování čar v obrázcích o celé body. Tento algoritmus je jednoduchý a je přímo implementován ve Photoshopu. Překážkou se stal až algoritmus pro zesilování čar o desetiny (setiny) bodu. Zde jsem vycházel z představy, že pro zesílení o desetinu bodu je potřeba obrázek 10x zvětšit, provést zesílení o jeden bod a obrázek zpět zmenšit. Postupně jsem zjistil, že je možné použít k reprezentaci pixelu matici a také, že ani tuto matici nemusím vyplňovat, protože obsahuje oblasti se shodnými hodnotami pixelů. Výsledná „jednoduchost“ algoritmu mě překvapila.

Dále jsem se snažil veškeré mnou navržené algoritmy zefektivnit. Myslím si, že v konečném výsledku se mi toto podařilo. Veškeré algoritmy bylo možné urychlit díky předzpracování obrázku, tj. díky používání již jednou vypočtených hodnot. Využití předzpracování vedlo nejen k zefektivnění algoritmů, ale i k jejich menší paměťové náročnosti.

Největší překážkou při implementaci algoritmů pro mě byla malá znalost vývojového prostředí Microsoft Visual C++ a téměř žádná znalost práce s Win32Api. Po prostudování dostupné literatury a po seznámení se s odbornými publikacemi na toto téma se mi podařilo porozumět problematice tvorby dialogů v prostředí Win32Api v takové míře, abych mohl navrhnout dialog a zpracovat obsluhu událostí od kontrol, které jsou na něm umístěné.

Pro pochopení problematiky zadané práce bylo nutné seznámit se s tvorbou pluginů pro Adobe Photoshop. I zde se jednalo o oblast, která byla pro mne nová a kterou jsem musel prostudovat a seznámit se s ní. Rád bych zde poděkoval panu ing. Jiřímu Skálovi za jeho publikaci „Tvorba zásuvných modulů pro Adobe Photoshop“, ze které jsem mohl čerpat a která mi velice pomohla při tvorbě cvičných pluginů a následně i při tvorbě programové části zadané práce.

Výsledkem programové části mé bakalářské práce je plugin pro Adobe Photoshop, který je schopen provádět pokročilé editace perokreseb. Obrázky, se kterými umí plugin pracovat, musí být v odstínech šedi. Každý odstín šedi musí být zobrazitelný na 8 bitech. Při

editaci může uživatel provést zesílení / ztenčení linek v perokresbě a následně provést dithering.

V případě další úpravy pluginu by bylo vhodné doplnit náhled o možnost přiblížení a oddálení části obrazu. V případě zesilování linek o desetiny (setiny) bodu se v náhledu sice změny projeví, jsou však při zobrazení špatně viditelné. Dále by bylo vhodné implementovat nějaký „lepší“ algoritmus detekce pozadí, aby uživatel nemusel řešit zesílení čar v okolí rozhraní mezi pozadími. Zde vidím slabinu současné implementace.

Použitá literatura

- [1] SKÁLA, Jiří. Tvorba zásuvných modulů pro Adobe Photoshop. Studentský sborník KIV FAV ZČU. Plzeň: KIV, 2006.
- [2] ŽÁRA, J., BENEŠ, B., SOCHOR, J., FELKEL, P. *Moderní počítačová grafika*. 2. vydání. Brno: Computer Press, c2004. ISBN 80-251-0454-0.
- [3] PETZOLD, Charles. *Programování ve Windows*. 1. Vydání. Praha: Computer Press, c1999. ISBN 80-7226-206-8