UNIVERZITA PARDUBICE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

GEOMETRICKÉ TRANSFORMACE, LINEÁRNÍ A NELINEÁRNÍ FILTRY

Bc. Luboš Hovorka

Diplomová práce
2009

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra softwarových technologií
Akademický rok: **2008/2009**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Luboš HOVORKA**

Studijní program: **N2646 Informační technologie**

Studijní obor: **Informační technologie**

Název tématu: **Geometrické transformace, lineární a nelineární filtrace v obraze**

Zásady pro vypracování:

- V úvodu práce je nutné provést přehled zadané problematiky (typy geometrických transformací včetně interpolačních algoritmů, druhy lineárních a nelineárních filtrů a jejich matematická interpretace, vysvětlení pojmů konvoluce, korelace atd.) a její kategorizace. - Hlavním cílem diplomové práce je realizace vzorových implementací výše uvedených algoritmů a filtrů pomocí vybraného vyššího programovacího jazyka, jejich posouzení a doporučení pro konkrétní nasazení v oblasti predzpracování obrazu. - Pro účely testování zkoumaných algoritmů a metod se využijí reálná obrazová data.

Rozsah grafických prací:

Rozsah pracovní zprávy:

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

Seznam odborné literatury:

Hlaváč V., Kybic J.: Geometrická transformace, ČVUT FEL Praha Castleman K. R.: Digital Image Processing, Prentice Hall, New York 1996. Dobeš M.: Zpracování obrazu a algoritmy v C, BEN - technická literatura, 2008

Vedoucí diplomové práce: **Ing. Zdeněk Šilar**
Katedra informačních technologií

Datum zadání diplomové práce: **31. října 2008**

Termín odevzdání diplomové práce: **22. května 2009**

L.S.

doc. Ing. Simeon Karamazov, Dr.
děkan

doc. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 4. listopadu 2008

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Souhlasím s prezenčním zpřístupněním své práce v Univerzitní knihovně.

V Pardubicích dne 28. 5. 2009

Luboš Hovorka

Poděkování

Touto cestou bych rád poděkoval mému vedoucímu práce panu Ing. Zdeňku Šilarovi za pomoc a trpělivost při řešení mé diplomové práce. Také rodině, která mi připravila podmínky k úspěšnému dokončení studia.

ANOTACE

Práce je zaměřena na implementaci metod předzpracování obrazu, které slouží k odstranění vad z obrazu. V teoretické části jsou popsány principy jednotlivých metod s ukázkami zpracovávaných obrazů. Výstupem práce je implementace vybraných metod, v programovacím jazyce C#, určených pro podporu výuky předmětů Počítačová grafika a Zpracování obrazu. V závěru jsou výsledky metod porovnávány a doporučovány pro konkrétní nasazení v oblasti předzpracování obrazu.

Klíčová slova

Zpracování obrazu; předzpracování obrazu; jasové transformace; operace s histogramem; odstranění šumu; detekce hran.

ANNOTATION

The thesis is aimed at the implementation of image pre-processing methods which serve the noise reduction from the picture. In the theoretical part, I describe the principles of particular methods with samples of images being processed. The output of the thesis is the implementation of selected methods in programming language C# designated for supporting teaching the subjects Computer Graphics and Image Processing. In conclusion, I compare and recommend the results of the methods for the concrete application in the section of image pre-processing.

KEYWORDS

Image processing; image pre-processing; intensity transformation; histogram operations; noise reduction; edge detection

Obsah

Seznam obrázků

Seznam tabulek

Seznam algoritmů

1 Úvod

Zpracování obrazu je kapitola, která se začala vyvíjet v 60. letech našeho století. Postupně s vývojem výpočetní techniky se oblast zpracování obrazu stále rozšiřovala a zasahovala do více a více oblastí. Nyní je zpracování obrazu považováno za nejatraktivnější oblast podnikání. Díky rozšíření digitální techniky, zasahuje do nejrůznějších odvětví průmyslu, kde poskytují životně důležité informace. Příkladem, kde se bez zpracování obrazu neobejdou, je oblast medicíny. Stále vyvýjené techniky zpracování obrazu, na kterých se podílí nespočet firem, posunuje zdravotnictví stále ku předu. Armáda, meteorologie, kartografie, počítačová grafika, strojírenství a mnoho dalších odvětví využívá těchto technik pro zvýšení efektivity práce.

Cílem této práce je vytvořit aplikaci, která bude koncipována jako software pro podporu výuky počítačové grafiky a zpracování obrazu. V aplikaci budou implementovány algoritmy předzpracování obrazu, které budou schopné provádět operace nad reálnými daty. Z důvodu obsáhlosti tématu, není možné implementovat všechny techniky předzpracování obrazu, proto si kladu za cíl vytvořit aplikaci s podporou modulů pro jednoduché rozšiřování.

V práci bude postupně probráno:

- Úvod do zpracování obrazu.
- Přehled metod předzpracování obrazu.
- Popis s ukázkami vytvořené aplikace.
- Dosažené výsledky porovnávané s výsledky programu Matlab.
- Závěry a doporučení jednotlivých metod předzpracování obrazu.

2 Zpracování obrazu

Cílem zpracování obrazu je získávat obrazové informace o reálném světe, vyhodnotit a porozumět jeho obsahu. Obrazové informace jsou nejčastěji snímány různými druhy kamer či jinými snímacími zařízeními (například skenery), které jsou poté přeneseny, nejčastěji do počítačů, k dalšímu zpracovávání. Cílem zpracování obrazu má být porozumění jeho obsahu. Tento celý postup zpracovávání obrazu lze kategorizovat do několika základních skupin:

- Pořízení a digitalizace obrazu,
- předzpracování obrazu,
- segmentace obrazu,
- porozumění obsahu obrazu.

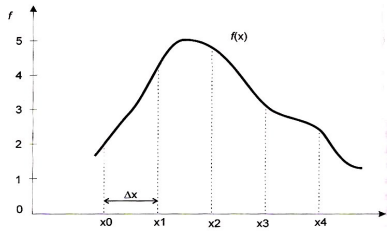
2.1 Pořízení obrazu

Prvním krokem zpracování obrazu je snímání obrazu, na které navazuje proces digitalizace. Typické snímače obrazu jsou různé druhy kamer, fotoaparátů, skenerů, které mají v sobě zabudované obrazové senzory (CCD, CMOS). Tyto senzory sejmou obraz a výstupem je nejčastěji spojitý signál. Abychom takto získaný obraz mohli dále počítačově zpracovávat, je ho nutné převést do diskrétní podoby. Tento proces převodu se nazývá *digitalizace* a probíhá ve dvou nezávislých krocích, jimiž jsou *vzorkování* a *kvantování*.

2.1.1 Vzorkování

Mějme spojitou funkci (viz. obr. 1), kterou jsme získali ze snímače. Vzorkování znamená zaznamenávání hodnot v určitých intervalech. Počátek vzorkování je zahájen v bodě x_0. Další vzorky jsou odebrány v diskrétním intervalu vzdáleném o Δx. Hodnotě Δx se říká vzorkovací interval. Převrácená hodnota vzorkovacího intervalu (vztah 1) udává frekvenci vzorkování.

$$f_{vz} = \frac{1}{\Delta x} \tag{1}$$



Obrázek 1 – Ukázka spojitého signálu získaného ze snímače obrazu (zdroj: [2])

Otázku určení vzdálenosti odebíraných vzorků řeší *Shannonova věta* o vzorkování, která říká, že signál lze rekonstruovat beze ztráty informace, pokud je vzorkováván s frekvencí minimálně dvojnásobnou než je maximální frekvence signálu obsaženého v obraze (vztah 2).

$$f_{vz} > 2f_{max} \tag{2}$$

V praxi je rozumné vzorkovat v co nejmenším intervalu, tj. pět krát větším, abychom nepřišli o žádné detaily v obraze. Překročíme-li však f_{max} vzniknou v obraze nežádoucí informace, které se v originálu nezobrazovali.

2.1.2 Kvantování

Další součást procesu digitalizace je kvantování. Kdy je nutné každému vzorku $x_0, \ldots, x_n$ přiřadit diskrétní hodnotu. Často tato hodnota je neceločíselná, proto je nutné tuto hodnotu zaokrouhlit k zisku celočíselné hodnoty, která v počítačové grafice představuje hodnotu jasu pixelu. Na obrázku 1 bude tedy vzorku x_0 přiřazena hodnota 2, $x_1 = 4$, atd. Z toho lze vyvodit, že každý odebraný vzorek odpovídá jednomu pixelu obrazu s hodnotou jasu získanou kvantování. Více o procesu digitalizace lze nalézt například v knihách [4], [1].

2.2 Předzpracování obrazu

Po úspěšném pořízení obrazu a jeho digitalizaci máme připraven digitální obraz určen k dalšímu zpracování. Obraz však může být poškozen. Chyby mohou vznikat díky způsobu snímání nebo také díky nevhodným okolním podmínkám

v průběhu snímání. Metody předzpracování obrazu obsahují mnoho různých metod na odstranění těchto chyb, které jsou schopny odstranit nebo alespoň výrazně potlačit. Výstup předzpracování je obraz, který je předán k dalšímu zpracovávání a to k *segmentaci obrazu*. Základní rozdělení metod předzpracování obrazu je:

- Jasové transformace,
- transformace založené na změnách histogramu,
- geometrické transformace,
- filtrace a detekce hran.

2.3 Segmentace

Nejtěžším krokem ve zpracování obrazu je právě část nazvaná segmentace obrazu. Pod pojmem segmentace obrazu se rozumí rozdělení obsahu na části, které nás zajímají a které ne. Výsledkem segmentace by měli být oblasti obrazu, které odpovídají objektům ve vstupním obraze. V tomto případě se jedná o tzv. úplnou segmentaci. Pokud vybrané oblasti neodpovídají přesně objektům vstupního obrazu, pak hovoříme o částečné segmentaci. Toto je jen základní popis segmentace obrazu. Více informací lze nalézt například [2], [3].

3 Předzpracování obrazu

Po pořízení obrazu určeného k další zpracování může na obraz působit mnoho negativních faktorů. Jako například špatná expozice při pořizování snímku fotoaparátem, užití špatné metody snímání, špatné okolní podmínky při snímání nebo také vady na snímačích. Tyto vady v obraze zabraňují správnému zpracování obrazu, proto je vhodné před finálovém zpracování vady odstranit. To je možné díky existujícím metodám předzpracování obrazu jako jsou jasové transformace, transformace založené na změnách histogramu, geometrické transformace a metody filtrace. Protože je tato kapitola příliš obsáhlá, jsou zde uvedeny jen některé vybrané metody a to pouze v prostorové doméně.

4 Jasové trasformace

Jasové transformace jsou v počítačové grafice obecně známé, proto zde stručně uvedu jen některé z nich. Jedná se o transformace u kterých není obraz předem analyzován jako u některých dalších uvedených transformací. Obraz se postupně prochází a na každém pixelu se provede předem daná operace.

4.1 Negativ

Ukázka operace negativ je na obrázku 2. Vytvoření negativu, pro 8bitový obraz, je dáno vztahem:

$$g(x, y) = 255 - f(x, y), \tag{3}$$

kde $f(x, y)$ je hodnota jasu vstupního obrazu, $g(x, y)$ je hodnota jasu výstupního obrazu a hodnota 255 je maximální intenzita 8bitového obrazu.

4.2 Gama korekce

Většina starších monitorů mají jednu nepříjemnou vlastnost a to, že mají nelineární jasovou odezvu intenzity fosforů na stínítku v závislosti na vstupním napětí katody. Křivka této nelineární závislosti zhruba odpovídá mocnině 2,5. Jinými slovy řečeno, pokud budeme chtít na monitoru zobrazit pixel s hodnotou jasu i, tak se ve skutečnosti zobrazí s hodnotou rovné $i^{2,5}$. Jelikož rozsah intenzity napětí se pohybuje v intervalu $\langle 0, 1\rangle$, znamená to, že zobrazovaná hodnota jasu bude menší než jakou byste čekaly ($0,5^{2,5} = 0,177$).

Gama korekce se využívá nejen pro opravu jasu na obrazovce, ale také pro opravu špatné expozice, kdy je požadovaná část objektu ukryta v příliš tmavé nebo naopak příliš světlé části obrazu. Gama korekce pro úpravu jasu je dána vztahem

$$y = y_{min} + (y_{max} - y_{min}) \cdot (\frac{x}{x_{max}})^{\frac{1}{\gamma}}, \tag{4}$$

kde y_{min}, y_{max} je rozsah jasu výstupního obrazu (často $\langle 0, 255\rangle$), x hodnota jasu vstupního obrazu, x_{max} maximální intenzita vstupního obrazu a γ koeficient korekce.

Obrázek 2 – Ukázka převodu obrazu na negativ a) vstupní obraz v odstínech šedé, b) výstupní obraz negativ

Obrázek 3 – Ukázka gama korekce a) vstupní obraz v odstínech šedé, b) výstupní obraz po gama korekci s koeficientem $\gamma = 0,3$

4.3 Roztažení kontrastu

Metoda roztažení kontrastu (Contrast stretching) se využívá v případech, kdy je rozdíl mezi různými úrovněmi jasu v obrazu příliš malý. Takový obrázek je nekontrastní a pro nás zajímavá informace se nachází v příliš úzkém pásmu jasových úrovní. Právě na tyto případy se aplikuje metoda roztažení kontrastu. Na motivačním obrázku 4 je nekontrastní obraz střech domů sejmutý ze satelitu. Po aplikaci transformace roztažení kontrastu je obraz opět kontrastní a jsou krásně rozpoznatelné detaily jednotlivých střech domů.

Prvním krokem je určení intervalu hodnot jasu $\langle a, b\rangle$, na který má být obraz roztažen. Pro standardní 8bitový obrázek jsou tyto limity nastaveny na hodnotu $\langle 0, 255\rangle$. Dalším krokem je určení rozsahu hodnot jasu $\langle c, d\rangle$ z histogramu vstupního obrazu. Pokud rozsah $\langle c, d\rangle$ je dostatečný, tj. je větší nebo roven rozsahu $\langle a, b\rangle$,

Obrázek 4 – Metoda roztažení obrazu (zdroj: [21]) a) vstupní obraz, b) modifikovaný obraz

neproběhne žádná transformace. Jinak se prochází vstupní obraz pixel po pixelu a na vstupní pixely r je aplikováno roztažení hodnot jasu dané vztahem

$$v = (r - c)\frac{b - a}{d - c} + a, \tag{5}$$

které se ukládá do pixelů v výstupního obrazu.

5 Transformace založené na změnách histogramu

5.1 Histogram

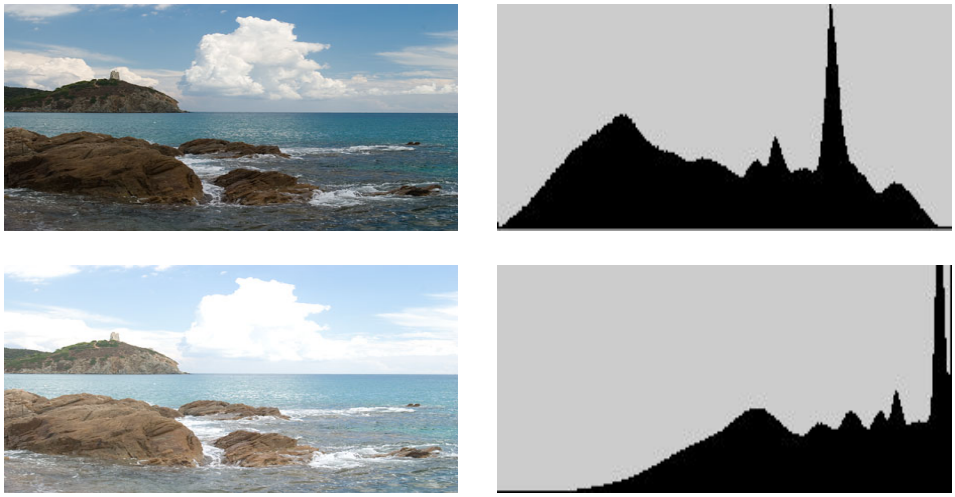
Nejjednodušší popis obrazu je histogram, který nám poskytuje informaci o četnosti pixelů v jednotlivých úrovní jasu obrazu. Nenese žádnou informaci o polohách pixelů, lze však z něj vyčíst jiné důležité informace o obraze.

Histogram je důležitou pomůckou v digitální fotografii. S nadsázkou se dá říct, že dobrý fotograf první na co kouká, po pořízení fotografie, je histogram, ze kterého je schopen vyčíst informace zda je obraz dobře pořízený (např. jasově vyrovnaný, dobrý kontrast, dobrá expozice). Až poté se podívá na celkový náhled fotografie. Velmi podrobně popsaný druhy histogramů lze nalézt v článku [19]. Ukázky histogramů barevného obrazu jsou na obrázku 5. První histogram ukazuje správně exponovaný obraz. Histogram začíná v bodě nula, tj. $[0, 0]$, poté stoupá, může mít jakýkoliv průběh a opět končí v bodě nula ($[255, 0]$). Dále také je vidět, že v obraze nejsou žádné velké plochy černé barvy (podpaly) ani plochy bílé barvy (přepaly). Z pohledu na histogram můžeme říct, že obraz je dobře pořízený. V druhé ukázce je přeexponovaný obraz. Obsahuje mnoho pixelů bílé barvy (přepaly), což může působit velmi rušivě.

Histogram se konstruuje tak, že na horizontální ose x se nacházejí tmavé odstíny blízko nulové hodnoty a s rostoucí hodnotou osy se intenzita zvyšuje. Na konci osy x se nacházejí nejvyšší intenzity odpovídající bílé barvě. Na vertikální osu jsou poté vyneseny četnosti pixelů jednotlivých úrovní jasu vstupního obrazu. Šedotónový obraz má pouze jeden histogram, odpovídající hodnotám jasu v obrazu. V barevném obraze modelu RGB bude každé složce odpovídat jeden histogram.

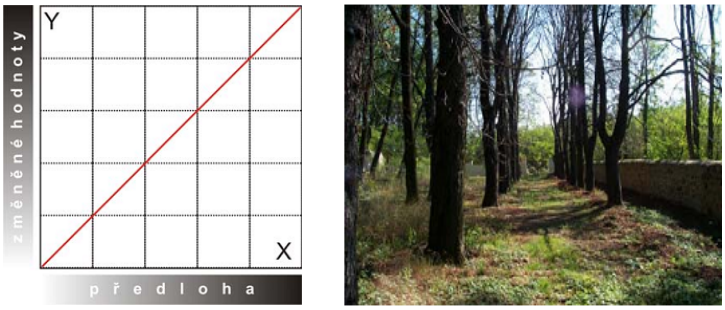
5.2 Změny histogramu

Operace, které mají za svůj následek změnu histogramu, bývá zvykem označovat jako operace s histogramem. I když editační programy zobrazují tyto operace jako přímé zásahy do histogramu, ve skutečnosti se jedná o operace, jejichž



Obrázek 5 – Ukázky obrazů s histogramy (zdroj: [20]) a) spávně exponovaný obraz, b) histogram správně exponovaného obrazu, c) podexponovaný obraz, d) histogram špatně exponovaného obrazu

výsledkem jsou změny histogramu. Nejčastěji se manipulace s jednotlivými složkami provádějí pomocí editačních křivek (viz. obr. 6). Tyto křivky přiřazují určité vstupní hodnotě jasu (0–255 na ose x) výstupní hodnotu jasu(0–255 na ose y) a nejčastěji se reprezentují pomocí jednorozměrné tabulky. Z tohoto důvodu se jim říká vyhledávací tabulka (Look-up table). Při práci s křivkami můžeme pracovat s jednotlivými kanály zvlášť, nebo najednou [1].



Obrázek 6 – Editační křivka (zdroj: [19]) a) křivka v počáteční poloze, b) vstupní obraz

Na ose x je vynesen rozsah hodnot jasu vstupního obrazu a na vertikální ose je vynesen odpovídající rozsah výstupního obrazu. Postup kreslení křivky spočívá

v tom, že body křivky jsou počítány v každém bodě rozsahu jasu zvlášť. Při tomto postupu je naplněna vyhledávací tabulka, z které se poté jednoduchým způsobem změní výstupní hodnoty jasu obrazu. Pro výpočet křivky je možné použít kterékoliv z druhů křivek. Více informací o křivkách je možné nalézt v publikaci [17].

5.3 Změny jasu a kontrastu, prahování

Zvýšení nebo snížení jasu se snadno realizuje posunem křivky do příslušné pozice. Na obrázku 7 je výstupní obraz po aplikaci změny polohy editační křivky. Výsledný obraz má zvýšené hodnoty jasu. Pro snížení jasu obrazu, by křivka byla vypouklá opačným směrem, tj. směrem dolu.

Obrázek 7 – Úprava jasu pomocí editační křivky (zdroj: [19]) a) křivka pro zvýšení jasu, b) výsledný obraz

Změny kontrastu, tj. zvýšení či snížení kontrastu, se provádí pomocí takzvané s–křivky (shaped curve). S–křivka dostala jméno podle jejího tvaru zakrouceného do písmene S. Při zvyšování kontrastu se s–křivka posune tak, aby stínům přiřadil tmavší odstíny a světlům přidá vyšší intenzitu, to je ukázáno na obrázku 8. Opačný tvar s–křivky, který lze analogicky odvodit, má za následek snížení kontrastu obrazu.

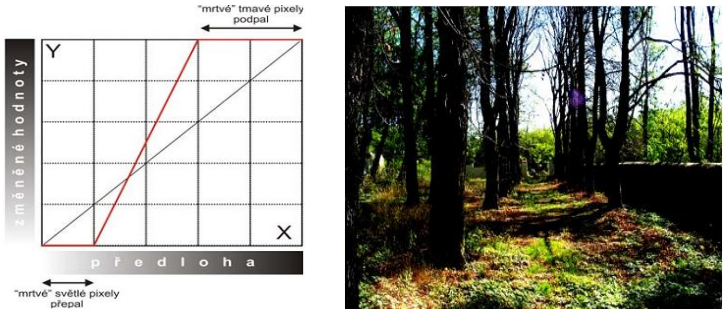
Pomocí křivek můžeme rovněž změnit bílí a černý bod v obraze. Stačí posunout minimální a maximální hodnotu směrem ke středu grafu. Tímto docílíme roztažení původního histogram jak je ukázáno na obrázku 9.

Obrázek 8 – Úprava kontrastu pomocí editační s–křivky (zdroj: [19]) a) křivka pro zvýšení jasu, b) výsledný obraz

Posunutím bílého a černého bodu obrazu na stejnou hodnotu, získáme operaci nazvanou *Prahování*. Spočívá v rozdělení jasové škály na dvě části. Každou takto získanou část nahradí jedinou hodnotou tak, jak je uvedeno v následujícím vztahu

$$i' = \begin{cases} L & pro\ i \leq T \\ H & pro\ i \geq T, \end{cases} \tag{6}$$

kde i je vstupní hodnota, i' je hodnota výstupní, T je práh a L a H jsou dvě výstupní hodnoty [1]. Všechny hodnoty s jasem menším než práh T jsou nahrazeny hodnotou s minimální intenzitou L a hodnoty větší než T jsou nahrazeny maximální intenzitou H. U 24bitového obrázku v RGB to znamená nahrazení $L = 0$ a maximální hodnotou $H = 255$.

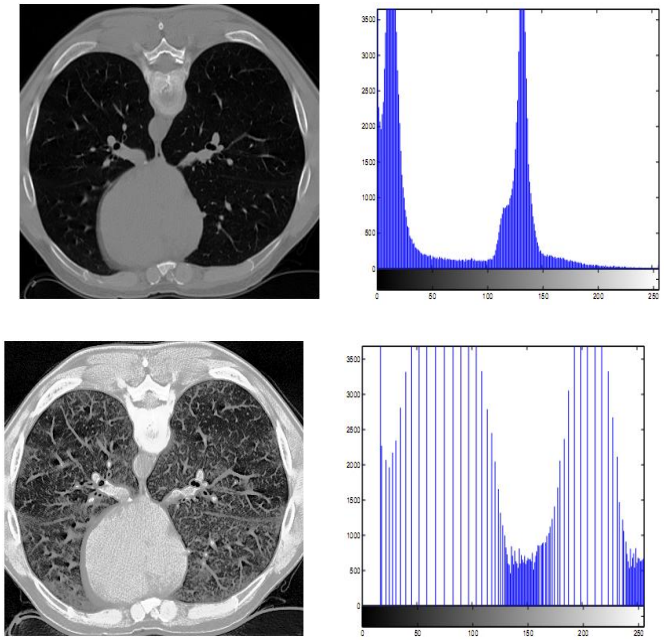


Obrázek 9 – Úprava bílého a černého bodu pomocí křivky (zdroj: [19]) a) křivka změny bílého a černého bodu, b) výsledný obraz

5.4 Ekvilizace histogramu

Metoda, která je často využívána při změně jasu obrazu se nazývá vyrovnání (ekvilizace) histogramu. Princip metody spočívá v rozložení jednotlivých jasových úrovní histogramu obrazu tak, aby každá úroveň byla zastoupena se stejnou četností.

Pokud získáme obraz, který je nekontrastní, tj. jednotlivé jasové složky se vyskytují nerovnoměrně, nebo je špatně exponovaný, je vhodné aplikovat tuto metodu pro vyrovnání jasu. Výsledkem bude kontrastní obraz s rozpoznatelnými detaily ukázaným na obrázku 10.



Obrázek 10 – Ekvilizace histogramu (zdroj: [4]) a) původní obraz, b) histogram původního obrazu, c) obraz po ekvilizaci, d) histogram po ekvilizaci

Postup ekvilizace histogramu je podrobně popsán,i s ukázkou algoritmu v knize [1]. Tak jen stručně popíši princip. Spočívá v nalezení takové *mapovací funkce*, jejíž aplikací na původní obraz získáme obraz s vyrovnaným histogramem. Ideálně vyrovnaný histogram, obsahuje všechny hodnoty zastoupené se stejnou četností, označme ji d. Hodnota d musí být průměrem ze všech hodnot a

počíta se dle vztahu 7.

$$d = \frac{width \times height}{max}, \tag{7}$$

kde $width$ je šířka obrazu, $height$ výška obrazu a max je maximální intenzita bodů v obraze.

Při vytváření mapovací se funkce chová tak, jako by bylo možno hodnotu vyšší než d rozdělit. Nejprve se prozkoumá do jakého intervalu je možné tuto hodnotu rozdělit a nerozdělenou ji potom umístí do jejího středu.

6 Geometrické transformace

Geometrické transformace jsou hojně využívány v počítačové grafice. Vypočítávají na základě souřadnic bodů ve vstupním obraze, souřadnice bodů ve výstupním obraze. V digitálním zpracování obrazu se navíc využívají ke korekci zdeformovaných obrazů (např. korekce geometrických vad objektivu kamery) nebo naopak k záměrnému deformování obrazu. Budu psát pouze o transformacích ve 2D prostoru, rozšíření do 3D je intuitivní.

Geometrická transformace obrazu je funkce T, která zobrazí bod (x, y) do bodu (x', y').

$$(x', y') = T(x, y) \tag{8}$$

Funkce T může být předem známa, jako je tomu v případě posunu, rotace, zvětšení, zkosení nebo operace vzniklé jejich složením, nebo je možné hledat transformační vztah na základě znalosti původního i transformovaného obrazu známé jako warping, kdy je zadána množina vztažných bodů mezi vstupním a výstupním obrazem a dle nich deformován. Jelikož je warping dost obsáhlé téma, nebudu se jím v této práci zabývat. Více informací o warpingu lze nalézt v publikacích [1], [11].

Geometrické transformace se skládá ze dvou kroků. Prvním krokem je *transformace souřadnic bodů* podle výše zmíněného vztahu. Druhým krokem je *interpolace jasu v obraze*, která hledá hodnotu jasu v celočíselné pozici původního obrazu počítané z neceločíselné pozice (x', y').

6.1 Transformace souřadnic bodů

Jednoduché transformace jsou dobře známy, tak jen ve stručnosti zmíním vztahy pro posunutí, otáčení, změnu měřítka a zkosení. Pro zjednodušení výpočtů se používá reprezentace bodů pomocí homogenních souřadnic. Velkou výhodnou homogenních souřadnic je, že vyjádření jednoduchých transformací docílíme pomocí jedné matice. To je také velká výhoda při skládání transformací, což si ukážeme později.

6.1.1 Posunutí

Posunutí (translace) o hodnotu t_x ve směru x a t_y ve směru y ja dáno vztahy

$$\begin{aligned} x' &= x + t_x, \\ y' &= y + t_y \end{aligned} \tag{9}$$

Matice transformace posunutí T má tvar

$$T(t_x, t_y) = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \tag{10}$$

6.1.2 Otáčení

Transformace otočení bodu kolem počátku souřadnic o úhel α je dáno soustavou

$$\begin{aligned} x' &= x\cos\alpha - y\sin\alpha, \\ y' &= y\sin\alpha + y\cos\alpha \end{aligned} \tag{11}$$

Matice transformace otáčení R má tvar

$$R(\alpha) = \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

6.1.3 Změna měřítka

Změna měřítka ovlivňuje současně polohu i velikost transformovaného objektu ve směru souřadnicových os. Pokud je absolutní hodnota měřítka v intervalu $< 0, 1 >$, dochází ke zmenšení a přiblížení transformovaného objektu k počátku souřadnic. Je-li větší než 1, dojde k prodloužení, je-li znaménko koeficientu záporné dochází k prodloužení či zmenšení v opačném směru. Změna měřítka s_x ve směru x a s_y ve směru y je dáno vztahy

$$\begin{aligned} x' &= s_x x, \\ y' &= s_y y \end{aligned} \tag{13}$$

Matice transformace změny měřítka S má tvar

$$S(s_x, s_y) = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{14}$$

6.1.4 Zkosení

Zkoseni sh_x ve směru x a sh_y ve směru y je dáno vztahy

$$\begin{aligned} x' &= x + sh_x y, \\ y' &= y + sh_y x \end{aligned} \tag{15}$$

Matice transformace zkosení SH_x a matice SH_y mají tvar

$$SH_x(sh_x) = \begin{bmatrix} 1 & sh_x & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad SH_y(sh_y) = \begin{bmatrix} 1 & 0 & 0 \\ sh_y & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{16}$$

6.1.5 Skládání transformací

Pokud například chceme otáčet bodem P, který neleží v počátku souřadnic, je třeba provést složenou transformaci. Tato transformace bude složena posunutí do počátku souřadného systému, otočení o požadovaný úhel a následné posunutí zpět na své počáteční místo. Takto vzniklou transformaci VT lze vyjádřit pouze jedinou maticí, která vznikne vynásobením všech matic jednotlivých transformací. V našem příkladě bude vypadat

$$VT = T(x_t, y_t) \cdot R(\alpha) \cdot T(-x_t, -y_t). \tag{17}$$

Výsledná transformace bodu P do P' bude vypadat

$$P' = VT \cdot P \tag{18}$$

6.2 Interpolace jasu v obraze

Druhým krokem po geometrické transformaci je interpolace jasu, kde se hledá hodnota jasu každého transformovaného bodu. Proto se pro každý bod transformovaného obrazu vypočítává odpovídající bod v původním obrazu pomocí transformačních funkcí. Z hodnot jasu původního obrazu se určí, jaká hodnota jasu odpovídá dané pozici výstupního bodu. Jelikož po transformaci (např. změna měřítka) souřadnice (x', y') již celočíselné být nemusí, musí se odpovídající hodnota určit interpolací.

Postup výpočtu při změně měřítka je uveden na obrázku 11. Pro každý bod výstupního obrazu B se vypočítají odpovídající souřadnice bodu v obrazu A pomocí invertováním transformačních funkcí T dané vztahem

$$(x, y) = T^{-1}(x', y'), \tag{19}$$

kde (x', y') jsou souřadnice výstupního obrazu a (x, y) souřadnice vstupního obrazu. Protože vypočtené souřadnice bývají nejčastěji neceločíselné hodnoty, musí se výsledná hodnota jasu určit interpolací. Výsledkem interpolace je hodnota jasu $f_n(x, y)$, kde index n rozlišuje různé typy interpolačních metod. Jas lze vyjádřit jako výpočet dvojrozměrné konvoluce, o které se zmiňuji v kapitole 7.1, dané vztahem

$$f_n(x, y) = \sum_{s=-\infty}^{\infty} \sum_{t=-\infty}^{\infty} f(x - s, y - t) h_n(s, t), \tag{20}$$

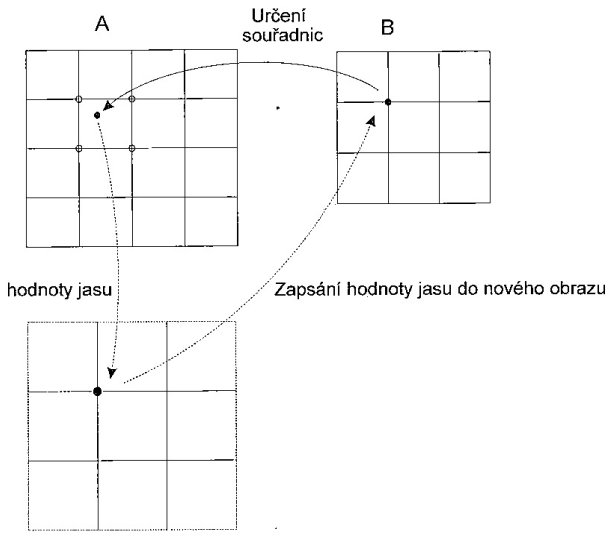
kde funkce f je vstupní obraz, funkci h_n se říká interpolační jádro.

Nejčastější metody, které se používají v počítačové grafice, jsou interpolace nejbližším sousedem, lineární, bilineární a kubická interpolace.

6.2.1 Nejbližší soused

Nejjednodušší metoda je interpolace nejbližším sousedem, kdy hodnota výsledného jasu je dána bodem vstupního obrazu, který je nejblíže k vypočítaným souřadnicím. K výpočtu nejbližšího souseda se využívá prostého zaokrouhlování souřadnic bodu dané vztahem

$$(x, y) = f(round(x), round(y)). \tag{21}$$



Obrázek 11 – Postup interpolace jasu (zdroj: [2])

Největší chyba bodu po interpolaci je nejvýše půl pixelu, která je zřetelná ve výsledném obraze. V obraze se chyby projevují na hranách s malým sklonem, při zvětšení zvýrazňuje skoky a při zmenšení poškozuje tenké čáry. Naopak velká výhoda je její rychlost. Konvoluční jádro interpolace má tvar

$$h(x) = \begin{cases} 0 & pro: \ \ 0 \leq |x| < 0,5 \\ 1 & pro: \ \ 0,5 \leq |x| \end{cases} \tag{22}$$

6.2.2 Bilineární interpolace

Bilineární interpolace využívá okolí svých čtyř sousedů k výpočtu hodnoty jasu, tak jak je ukázáno na obrázku 12. Známé hodnoty pixelů jsou $Q_{11} = (x_1, y_1)$, $Q_{12} = (x_1, y_2)$, $Q_{21} = (x_2, y_1)$, $Q_{22} = (x_2, y_2)$, díky kterým budeme schopny vypočítat hledanou hodnotu jasu v bodě P. Jelikož je bilineární interpolace separabilní, je možné hledání hodnoty v bodě P rozdělit do dvou kroků. Nejprve aplikovat lineární interpolaci na řádky a poté na sloupce obrazu. To znamená, že bod $R_1 = (x, y_1)$ bude nejprve vypočten ze sousedních bodů Q_{11} a Q_{21} a bod $R_2 = (x, y2)$ z bodů Q_{12} a Q_{22} a to dle vztahů

$$f(R_1) = \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x - x_1}{x_2 - x_1} f(Q_{21}) \tag{23}$$

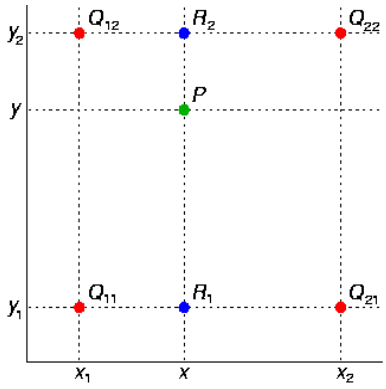
$$f(R_2) = \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x - x_1}{x_2 - x_1} f(Q_{22})$$

Po výpočtu ve směru osy x je nutné provést finální výpočet a to ve směru osy y a dopočítat tak hodnotu hledaného bodu P, který je dán vztahem

$$f(P) = \frac{y_2 - y}{y_2 - y_1} f(R_1) + \frac{y - y_1}{y_2 - y_1} f(R_2). \tag{24}$$

Konvoluční jádro pro lineární interpolaci se nazývá *tent function* a má tvar

$$h(x) = \begin{cases} 1 - |x| & pro: \ 0 \leq |x| < 1 \\ 0 & pro: \ 1 \leq |x| \end{cases} \tag{25}$$

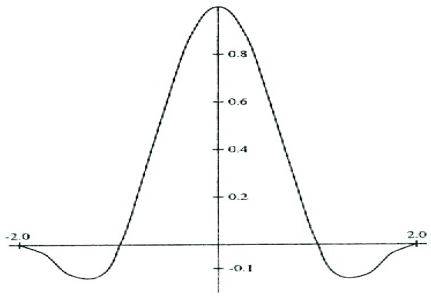


Obrázek 12 – Bilineární interpolace (zdroj: [23])

Bilineární interpolace je rychlá, neboť o proti jiným metodám využívá pouze čtyř sousedních bodů. Její nevýhodou je, že rozmazává původní ostré přechody v obraze.

6.2.3 Bikubická interpolace

Lepších výsledků, než v předchozích případech, dosahuje bikubická interpolace, která využívá funkce s hladším průběhem (viz obr. 13). Pro své výpočty potřebuje mnohem více sousedních pixelů, přesně šestnáct, než předchozí metody a jsou proto výpočetně náročnější.



Obrázek 13 – Tvar průběhu funkce jádra bikubické interpolace (zdroj: [1])

Konvoluční jádro používané pro bikubickou interpolaci má tvar

$$h(x) = \begin{cases} |x|^3 - 2|x|^2 + |x| & pro: \ 0 \leq |x| < 1 \\ -|x|^3 + 5|x|^2 - 8|x| + 4 & pro: \ 1 \leq |x| < 2 \\ 0 & pro: \ 2 \leq |x| \end{cases} \tag{26}$$

Jinou často používanou metodou pro výpočet bikubické interpolace je interpolace pomocí kubické B–spline křivky, která má tvar jádra danou vztahem

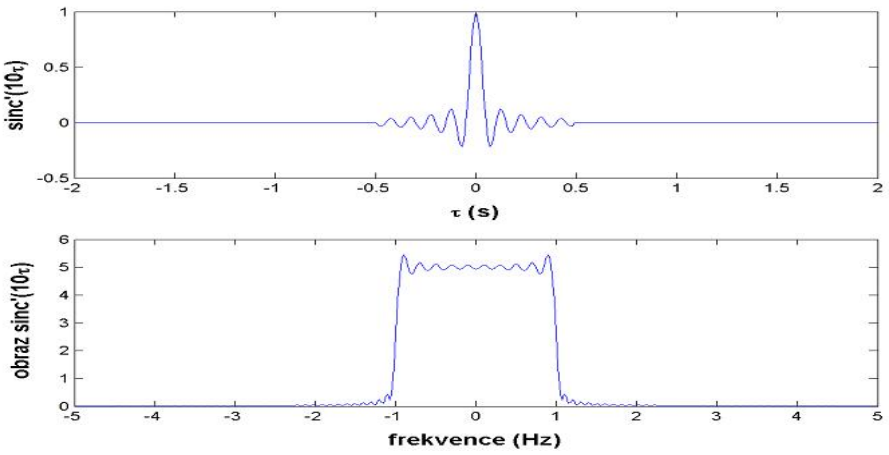
$$h(x) = \frac{1}{6} \begin{cases} 3|x|^3 - 6|x|^2 + 4 & pro: \ 0 \leq |x| < 1 \\ -|x|^3 + 6|x|^2 - 12|x| + 8 & pro: \ 1 \leq |x| < 2 \\ 0 & pro: \ 2 \leq |x| \end{cases} \tag{27}$$

Toto jádro interpolace se též nazývá Parzenovo okno [1].

6.2.4 Sinc

Sinc filtry byly navrženy pro co nejpřesnější rekonstrukci obrazu. Odvození vychází ze spojitých signálů, které jsou rekonstruovány s co největší přesností. Protože odvození je příliš složité a jsou potřeba velké znalosti z matematiky, nebudu odvození ukazovat, pouze odkáži na literaturu [18], ve které je možné tyto informace nalézt. Nás bude zajímat pouze to, že na základě myšlenky ideální rekonstrukce pomocí funkce $sinc$ (viz. obr. 14), která je daná vztahm

$$sinc(x) = \frac{sin(\pi x)}{\pi x} \tag{28}$$



Obrázek 14 – Funkce $sinc'$ a její Fourierův obraz (zdroj: [18])

a má tvar obdélníku, je založena celá řada metod pro převzorkování. Tyto metody se snaží funkci $sinc$ s omezeným definičním oborem nahradit jinou funkcí (tzv. konvoluční jádro), jejíž obraz by se nejvíce podobal ideálnímu obdélníkovému tvaru. Jedním ze způsobů jak funkci vytvořit je součin původní funkce $sinc(x)$ a vhodně zvolené funkce, která se nazývá okno. Tím vznikne námi požadované konvoluční jádro, které nahrazuje ideální $sinc(x)$ funkci.

6.2.4.1 Lanczosův filtr

Lanzcosův filtr je pojmenován po maďarské matematikovi a fyzikovi Corneliu Lanczosovi. Stal se velmi populární díky své dobré kvalitě výsledků a dostatečné výpočetní rychlosti. Výsledné konvoluční jádro vznikne vynásobením Lanczosova okna dané vztahem

$$LanczosOkno(x) = \begin{cases} \frac{\sin(\pi x/N)}{\pi x/N} & pro: \quad |x| \leq N \\ 0 & pro: \quad |x| > N \end{cases} \tag{29}$$

a funkce $sinc(x)$. Výsledné konvoluční jádro má tvar

$$LanczosFilter(x) = \begin{cases} \frac{\sin(\pi x/N)}{\pi x/N}\frac{\sin(\pi x)}{\pi x} & pro: \quad |x| \leq N \\ 0 & pro: \quad |x| > N \end{cases} \tag{30}$$

Lanczosův filtr není jediný možný filter. Jednotlivé filtry se liší šířkou okna. Šířka okna je dána hodnotou N, která nejčastěji nabývá hodnot $N = 2$, $N = 3$ a

nebo $N = 8$. Samozřejmostí je, že čím větší okno, tím lepší kvalita, bohužel na úkor času.

Existují i jiná často používaná okna jako například Hammingovo nebo Kaiserovo. Více o dalších užívaných oknech i s porovnáním výsledků lze nalézt v práci [24].

7 Filtrace

Tato kapitola patří metodám filtrace (předzpracování) obrazu, které k výpočtu nové hodnoty využívají okolí zpracovaného bodu obrazu. Okolí je tvořeno malým čtverec, který je posunován po obraze a jsou tak vypočteny pixely výstupního obrazu. Takový postup se nazývá *konvoluce*. Více o konvoluci se dovíte v kapitole 7.1.

Podle účelu se metody předzpracování, dle [4], rozdělují do dvou skupin. První skupina, *vyhlazování*, se snaží potlačit nežádoucí artefakty v obraze (např. šum, či jiné poškození obrazu). Tyto metody jsou příbuzné dolní propusti ve frekvenční oblasti. Do druhé skupiny patří detektory hran, které se snaží z hodnot v okolí reprezentativního pixelu nalézt hranu. Hrany v obraze odpovídají prudkým změnám hodnot jasu, které se dají detekovat pomocí výpočtu gradientů. Proto jsou tyto metody někdy označovány jako *gradientní metody*. Tyto dvě skupiny jsou navzájem protikladný, proto byli nalezeny nelineární metody, které vyhlazují a přitom se snaží zachovat nepoškozené hrany.

7.1 Konvoluce

Konvoluce je operace, která se velmi často používá v počítačové grafice. Spojitá konvoluce jednorozměrných funkcí $f(x)$ a $g(x)$ je definována vztahem

$$(f * g) = \int_{-\infty}^{\infty} g(\alpha) f(x - \alpha) d\alpha \tag{31}$$

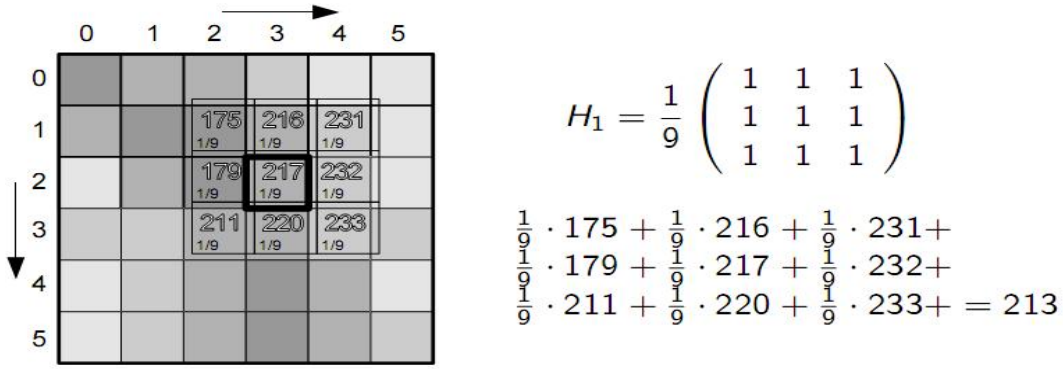
Funkci $g(x)$ se říká konvoluční jádro (*kernel*). Hodnota konvoluce funkce f s jádrem g v bodě x je integrál ze součinu funkce f s otočenou funkcí konvolučního jádra posunutou do bodu x [25].

Při zpracování obrazu bývá funkce $f(x)$ vstupní obraz, $g(x)$ maska (konvoluční jádro) a výsledek operace $(f * g)$ je výstupní obraz. Při počítačovém zpracování obrazu máme k dispozici navzorkovaný obraz s diskrétními hodnotami. Proto abychom mohli aplikovat metodu konvoluce na tento diskrétní obraz musíme vyjádřit konvoluci v diskrétním případě. To je provedeno pomocí sumy jako

aproximaci integrálu. Vzorec diskrétní konvoluce v 2D případě má tvar:

$$(f * g) = \sum_{s=-k}^{k} \sum_{t=-k}^{k} g(s,t) f(x-s, y-t) \tag{32}$$

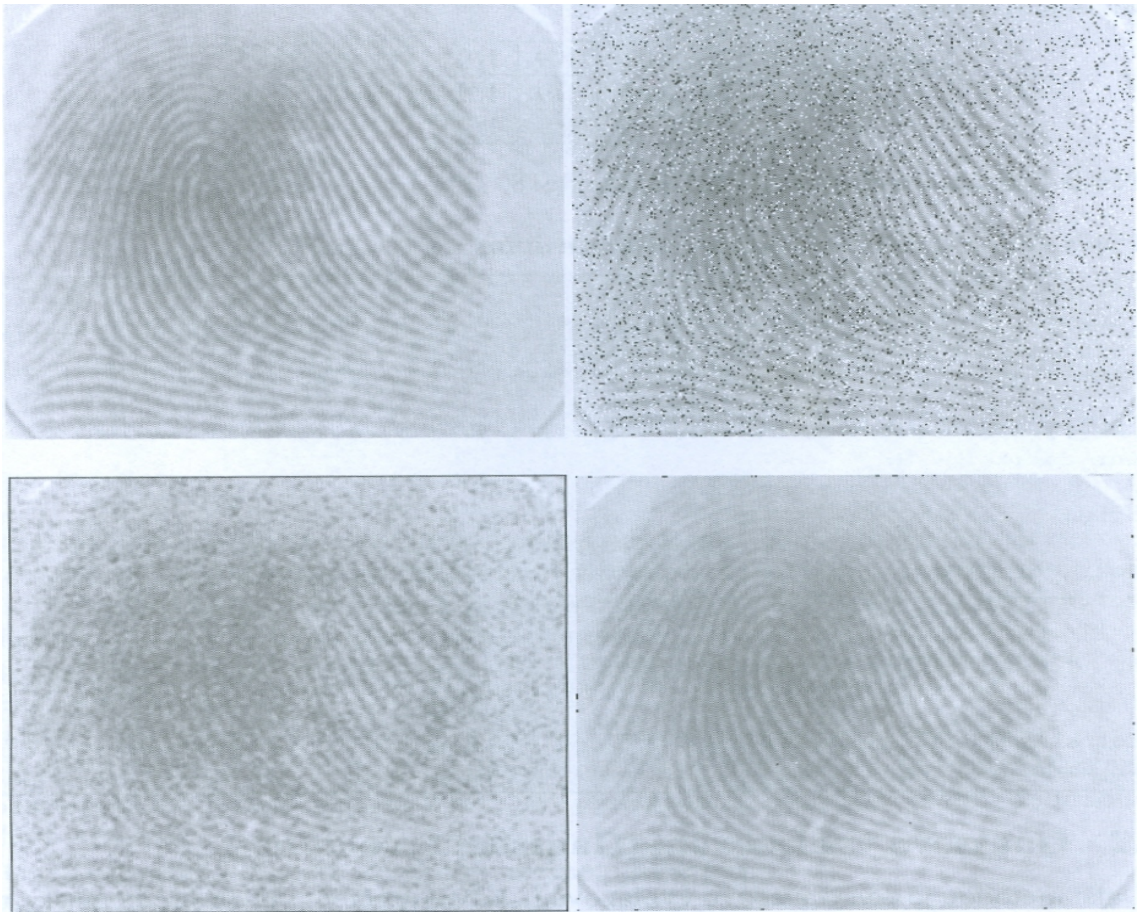
Princip metody spočívá v posunování masky po jednotlivých pixelech obrazu. Při každém posunutí masky se vypočítá součin koeficientů masky s hodnotou jasu v obraze pod maskou a spočítá se suma těchto součinů, která se vloží do výsledného obrazu na odpovídající pozici. Výsledná pozice se obvykle definuje vzhledem ke středu masky. Nejčastěji používané masky mají lichou velikost jak v řádcích, tak i v sloupcích. Ukázka masky $[3, 3]$ a postupu konvoluce je na obrázku 15.



Obrázek 15 – Postup konvoluce (zdroj: [8])

Při aplikaci masky, na pixely lemující obraz, dochází k překročení okraje obrazu. To je nutné vyřešit hned při návrhu algoritmu. Existuje několik možností řešení a jsou to:

- Obraz se zvětší o polovinu masky a tyto nové hodnoty se naplní nulami, které zařídí ignorování hodnot za okraji obrazu.
- Maska se pohybuje pouze v oblasti obrázku. U tohoto způsobu dochází k chybám na krajích obrazu, jelikož kraje nejsou přepočítávány.
- Na okrajích je možné použít zmenšenou masku (vhodné pouze pro některé případy).



Obrázek 16 – Výsledky různých způsobů filtrace (zdroj: [2]) a) originál, b) obraz zašumělý šumem typu pepř a sůl, c) výsledek vyhlazování průměrováním, d) výsledek mediánové filtrace

7.2 Lokální metody vyhlazování

Snímané obrazy často ovlivňují nežádoučí jevy. Příkladem těchto jevů může být šum, který se nejčastěji projevuje jako nepříjemné zrnění (viz obr. 16). Díky metodám vyhlazování obrazu je možné tento šum či nějaké jiné nežadoucí artefakty v obraze potlačit. Existuje mnoho různých druhů šumu, jejíž kategorizaci a popis lze nalézt například v knihách [1], [10], a pro každý typ je nutné vybrat tu metodu, která bude nejefektivnější, jelikož se mnohé mohou lišit.

7.2.1 Lineární metody vyhlazování

Lineární metody vyhlazování vypočítávají novou hodnotu aktuálního pixelu jako lineární kombinaci hodnot ze svého okolí. Pro obraz v digitální podobě lze v prostorové oblasti lineární kombinaci vyjádřit diskrétní konvolucí. Jednotlivé lineární filtry se liší hodnotami konvolučních masek g, které jsou aplikovány na obraz.

7.2.1.1 Vyhlazování průměrováním

Nejjednodušší metoda vyhlazování pomocí konvoluce je metoda průměrování, která každému bodu obrazu přiřadí novou hodnotu jasu, vypočtenou jako průměr jasů bodů z jeho okolí. Tím se snažíme eliminovat body, které mají větší nebo naopak menší intenzitu jasu. Maska 3×3 je zobrazena na obrázku 17, někdy je taktéž používána maska se zvětšenou vahou středního bodu. Tyto masky počítají výslednou hodnotu z okolních 8 pixelů a sebe sama. Nevýhodou této metody je, že rozmazávají hrany obrazu a proto nejsou vhodné k odstraňování šumu. Využívají se pouze pro vyhlazování obrazu.

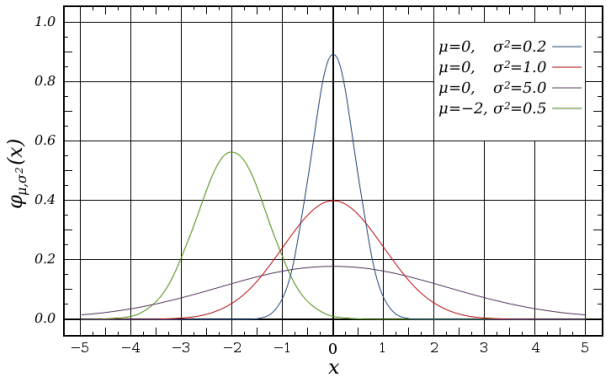
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

1/10	1/10	1/10
1/10	2/10	1/10
1/10	1/10	1/10

Obrázek 17 – Konvoluční masky pro metodu vyhlazování průměrováním

7.2.1.2 Guassovo vyhlazování

Další lineární metodou je tzv. Gaussovo vyhlazování, které se nejčastěji využívá k vyhlazování obrazu a odstranění Gaussova šumu. Konvoluční maska v jednorozměrném případě se vypočítává dle vztahu 33, kde m je střední hodnota (odpovídá středu masky $m = 0$) a σ^2 je rozptyl, která ovlivňuje strmost funkce (velikost masky). Koeficienty blíže středu masky mají vyšší váhu a odpovídají



Obrázek 18 – Příklady Gaussových křivek (zdroj: http://en.wikipedia.org)

hodnotám na Gaussově křivce zobrazené na obrázku 18.

$$h(x,y) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-m)^2}{2\sigma^2}} \tag{33}$$

Takto vypočítané masky mohou být příliš velké, to výrazně zvyšuje dobu výpočtu, proto je nutné velikost masky omezit. V praktickém použití, maska s rozptylem $\sigma > 3$ nemá smysl, pokud nepožadujeme opravdu velké rozmazání obrazu. Například ukázka masky pro $\sigma = 1$ má tvar

$$g(x) = \frac{1}{1000}[4, 54, 242, 400, 242, 54, 4].$$

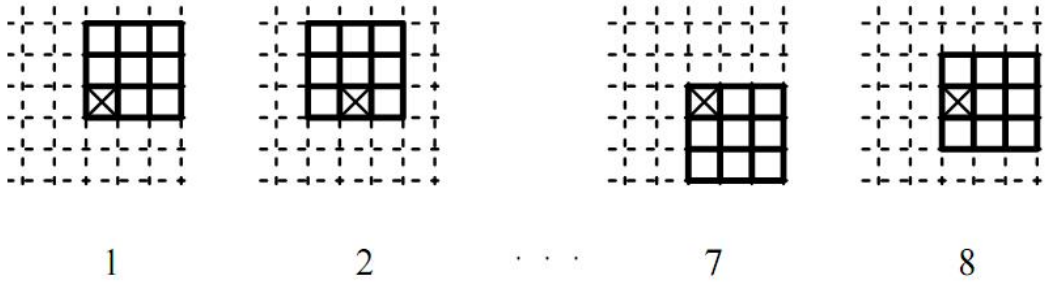
Hodnoty masky se musí normovat tak, aby součet hodnot masky byl roven jedné. Velkou výhodou, která zmenší časovou složitost algoritmu ja vlastnost nazvaná separabilita, díky které se zmenší složitost algoritmu. Nejprve se jednorozměrná maska aplikuje na řádky obrazu a poté na sloupce. Mějme obraz o rozměrech $N \times N$ a masku velikosti $M \times M$, pak složitost Gaussovy filtrace je $O(M^2N^2)$. Díky separabilitě je velikost masky $1 \times M$ a výsledná složitost je $O(MN^2)$.

7.2.2 Nelineární metody vyhlazování

Lineární metody mají jednu vadu a tou je, že rozmazávají hrany v obraze. Nelineární metody vyhlazování se snaží tuto chybu částečně eliminovat.

7.2.2.1 Metoda rotující masky

Metoda rotující masky je ukázána na obrázku 19. Principem metody je posunování masky kolem reprezentativního pixelu. Celkový počet masek je osm, kde pro každou polohu se vypočte rozptyl jasů celého okolí a vyhrává ta maska, která má rozptyl nejmenší. Z té je pak vypočítána hodnota jasu pomocí aritmetického průměru. Metoda nerozmazává hrany obrazu a má dokonce mírně ostřící charakter.



Obrázek 19 – Metoda rotující masky (zdroj: [4])

7.2.2.2 Geometrický vyhlazovací filtr

Geometrický vyhlazovaci filtr je dán výrazem

$$f(x,y) = [\prod_{(s,t)\in S_{xy}} g(x,t)]^{\frac{1}{mn}}. \tag{34}$$

V této metodě se pro každý pixel obrazu vypočítá součin hodnot pod maskou, který se poté umocní převracenou hodnotu $(m \times n)$, kde m,n jsou rozměry masky. Výsledná hodnota je poté uložena do výsledného obrazu na pozici (x, y). Tato metoda vykazuje porovnatelné výsledky jako metoda průměrování, pouze s tím rozdílem, že zachovává více detailů v obraze.

7.2.2.3 Harmonický vyhlazovací filtr

Harmonický vyhlazovací filtr je dán výrazem

$$f(x,y) = \frac{mn}{\sum_{(s,t)\in S_{xy}} \frac{1}{g(s,t)}}. \tag{35}$$

Metoda pracuje podobně jako geometrický filtr, s tím rozdílem, že je na každý pixel a jeho okolí aplikován jiný vzorec. Dobré výsledky dosahuje u šumu typu sůl, ale selhává u šumu typu pepř.

7.2.2.4 Metody s omezením jasu

Dalším typem nelineárního filtru je metoda obyčejného průměrování s omezením jasu. Tato metoda využívá obyčejného průměrování, ale brání rozmazání hran tím, že uloží výsledek průměrování pouze tehdy, je-li diference mezi hodnotou jasu původního bodu $f(x, y)$ a výsledkem průměrování p menší než zvolený práh $sigma$(viz vztah 36).

$$f(x,y) = \begin{cases} p & pro: \ |f(x,y) - pr| \leq sigma \\ f(x,y) & pro: \ |f(x,y) - p| > sigma \end{cases} \tag{36}$$

Podobným filtrem, který pracuje také s omezením jasu se nazývá *Sigma filtr*. Filtr počítá aritmetický průměr jen z těch okolích pixelů $g(x, y)$ pro které platí

$$|f(x,y) - g(x+i, y+j)| \leq sigma \tag{37}$$

7.2.2.5 Medián filtr

Mediánový filtr (*Median filter* nebo *Order-statistics*) je zástupce filtrů, který má původ v robustní statistice. Medián je prostřední prvek uspořádané posloupnosti hodnot. Při lichém počtu prvků je medián roven $f_{(n+1)/2}$. Při sudém počtu prvků je medián aritmetickým průměrem obou prostředních hodnot $f_{med} = (f_{n/2} + f_{(n/2)+1})/2$. Princip metody spočívá v posunování pomyslné masky po obraze a na každém bodu obrazu se vybere prostřední hodnota jasu z hodnot pod touto maskou. Výborné výsledky vykazuje u filtrace šumu typu sůl a pepř (viz. obr. 16), kde odstraní šum a zachová hrany v obraze.

7.2.2.6 Min a Max filtr

Tyto filtry mají stejný původ jako mediánový filtr. Vlastně pracují na stejné principu, pouze místo mediánu se vybírá nejmenší nebo největší hodnota jasu

z uspořádané posloupnosti hodnot. Min filtr vybírá tu nejmenší, Max filtr tu největší. Obecně se oba filtry používají na odstranění šumu typu sůl a pepř.

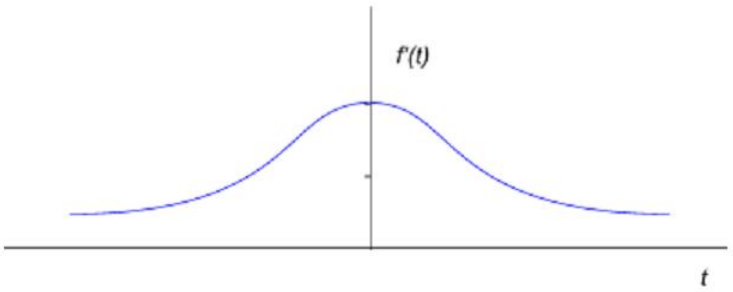
7.3 Detekce hran

Detekce hran je opět velmi rozsáhlá kapitola zpracování obrazu, ve které jde o nalezení takových míst v obrazu, kde dochází k prudké změně hodnot jasu. Tyto místa se nazývají hrany a jsou hledána pomocí hranových detektorů (*edge detector*). Nástrojem na hledání hran v obraze jsou parciální derivace. Pomocí derivace je nalezen gradient funkce, z kterého lze získat informace o velikosti gradientu (velikost změny funkce jasu) a směru gradientu (směr největší změny funkce jasu). Podle velikost gradientu je určeno zda se jedná o hranu či nikoliv. K výpočtům gradientů se užívají buď první nebo druhé derivace.

7.3.1 První derivace

Výsledek první derivace obrazu ve směrech x a y je gradient, který je dán vztahem

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial x} \end{bmatrix} \tag{38}$$



Obrázek 20 – První derivace funkce (zdroj: [16])

Podle hodnoty gradientu je určováno zda se o hranu jedná nebo ne. Půjde-li o hranu, bude výsledek derivace funkce největší jak ukazuje obrázek 20. Hodnota

gradientu funkce dvou proměnných $f(x, y)$ je definován jako vektor

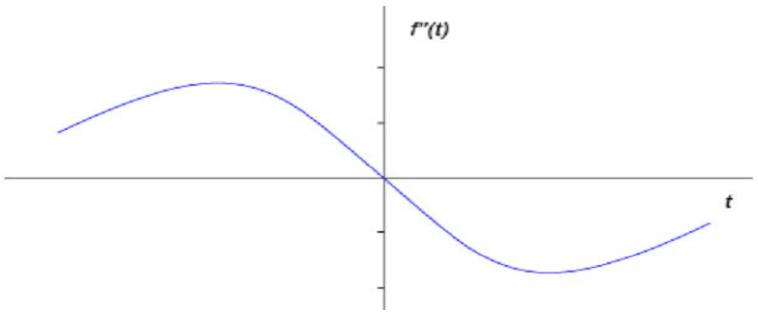
$$mag(\nabla f) = \sqrt{G_x^2 + G_x^2}. \tag{39}$$

Jelikož výpočet velikosti gradientu podle vztahu 39 je výpočetně náročný, je tento vzorec nahrazován zjednodušenou verzí a to vztahem

$$mag(\nabla f) = |G_x| + |G_x|. \tag{40}$$

K výpočtu gradientu se také často využívá konvoluce ve smyslu filtrování obrazu. Jsou navrženy konvoluční masky, které po aplikaci vrací hodnoty gradientu. Konvoluční maska se aplikuje nejprve v jednom směru x, která vrací gradient G_x. Poté se maska transponuje a je aplikována na směr y, z kterého získáme hodnotu G_y. Tabulka 1 ukazuje výčet některých filtrů s jejich konvolučními jádry.

7.3.2 Druhá derivace



Obrázek 21 – Druhá derivace funkce intezity (zdroj: [16])

Metody výpočtu druhé derivace jasové funkce jsou založeny na hledání druhé derivace procházející nulou, jak to můžete vidět na obrázku 21. Hledat polohu hrany v místě průchodu nulou je díky strmosti přechodu mnohem spolehlivější než hledání maxima u první derivace. Bohužel bude také více citlivější na šum. Druhou derivaci ve směru x lze v diskrétním obraze počítat jako rozdíl rozdílů hodnot jasu ležících vedle sebe dané vztahem

$$\begin{aligned} \frac{\nabla^2}{\nabla x^2} &\approx [f(x+1) - f(x)] - [f(x) - f(x-1)]] \\ &= [f(x+1) - 2f(x) + f(x-1)] \end{aligned} \tag{41}$$

Název filtru	G_x	G_y
Robertsův	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$
Prewitův	$\begin{bmatrix} 1 & 0 & -1 \\ 1 & 0 & -1 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$
Sobelův	$\begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$
Kirschův	$\begin{bmatrix} 3 & 3 & -5 \\ 3 & 0 & -5 \\ 3 & 3 & -5 \end{bmatrix}$	$\begin{bmatrix} -5 & -5 & -5 \\ 3 & 0 & 3 \\ 3 & 3 & 3 \end{bmatrix}$
Frei-Chenův	$\begin{bmatrix} 1 & 0 & -1 \\ \sqrt{2} & 0 & -\sqrt{2} \\ 1 & 0 & -1 \end{bmatrix}$	$\begin{bmatrix} -1 & -\sqrt{2} & -1 \\ 0 & 0 & 0 \\ 1 & \sqrt{2} & 1 \end{bmatrix}$

Tabulka 1 – Výčet konvolučních filtrů první derivace.

Z výsledku vztahu 42 zmíněného výše, získáme masku $[1, -2, 1]$ pro jednorozměrný případ ve směru x aplikovatelnou na tři sousední pixely. Pokud tuto masku rozšíříme na masku 3×3 a přičteme stejně rozšířenou masku pro jednorozměrný případ ve směru y, získáme konvoluční masku pro Laplaceův operátor, který aproximuje druhou derivaci funkce jasu (viz vztah 42).

$$\begin{bmatrix} 0 & 0 & 0 \\ 1 & -2 & 1 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{42}$$

Tímto způsobem můžeme získat i jiné typy masek, která mohou mít tvar

$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \qquad \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix}.$$

Ukázky aplikace Laplaceových operátorů jsou na obrázku 22.

Obrázek 22 – Ukázka aplikace různých Laplacových operátorů (zdroj: [9])

7.3.2.1 Laplacian of Gaussian (LoG)

Laplaceův operátor je ještě citlivější na šum než gradient, jednou cestou jak potlačit šum v obraze je vyhlazení. Tím docílíme odstranění slabých hran a také potlačení vysokých frekvencí. K vyhlazení obrazu se užívá Gaussovo vyhlazování zmíněné v kapitole 7.2.1.2. Po vyhlazení je aplikován Laplaceův operátor na detekci hran. Všechny takto nalezené hrany jsou významné, ty nevýznamné jsou potlačeny vyhlazováním.

K tomu abychom nemuseli nejprve počítat konvoluci Gaussovým operátorem a poté znovu konvoluci Laplaceovým operátorem, je možné spojit tyto dvě operace do jedné. Takto spojená oparace je označována jako LoG (*Laplacian of Gaussian*) a je dána vztahem

$$\nabla^2 G(x,y) = (\frac{x^2+y^2-\sigma^2}{\sigma^4})e^{-r^2/2\sigma^2}. \tag{43}$$

Tento vztah lze jednoduše odvodit, stačí vypočítat druhou derivaci Gaussova vyhlazování mající tvar

$$G(r) = e^{-\frac{r^2}{2\sigma^2}}, \tag{44}$$

kde $r^2 = x^2 + y^2$.

7.3.2.2 Diference z Gaussiánů

Jiný hranový detektor, který aproximuje LoG operátor, je DoG operátor (*Difference of Gaussian*). Tato metoda vypočítá dva Gaussově vyhlazené obrazy s různou hodnotou σ^2, které se poté od sebe odečtou. Celý proces DoG operátoru je dán vztahem

$$f(x,y) = \frac{1}{2\pi\sigma_1^2} e^{-\frac{(x^2+y^2)}{2\sigma_1^2}} - \frac{1}{2\pi\sigma_2^2} e^{-\frac{(x^2+y^2)}{2\sigma_2^2}}. \tag{45}$$

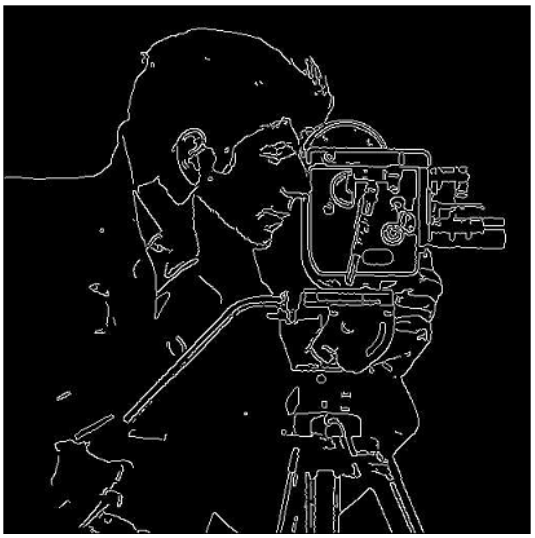
7.3.3 Cannyho hranový detektor

Jako nejoptimálnější hranový detektor je považován Cannyho hranový detektor. Je navržen tak, aby splňoval tři základní kritéria:

- **Kritérium detekce** Vyžaduje, aby významné hrany nebyli při hledání opomenuty a nesmí být detekovány vícekrát.
- **Kriterium lokalizace** Poloha detekované hrany musí být co nejpřesnější.
- **Kritérium jedné odezvy** Vyžaduje, aby na nalezené hrany byla jen jediná odezva. Toto částečně zajišťuje již první kritérium. Tento případ je zaměřen hlavně na zašuměné a nehladké hrany.

Díky těmto požadavkům na hranový detektor, se stal nejpopulárnější. Jiné detektory nedosahují tak kvalitních výsledků jako právě Cannyho hranový detektor. Konstrukce Cannyho detektoru se dá rozdělit do několika kroků. Prvním krokem je vyhlazení obrazu. To je prováděno pomocí Gaussova vyhlazování (viz kapitola 7.2.1.2), které odstraní nepodstatné hrany a eliminuje šum v obraze. U Gaussova vyhlazování platí, že čím větší velikost masky, tím bude citlivost na šum v obraze menší.

Po vyhlazení obrazu je dalším krokem nalezení hran. To se často realizuje nejčastěji Sobelovým detektorem. O Sobelovém detektoru se dočtete v kapitole 7.3.1. Podstatou je nalezení gradientů ve dvou směrech G_x a G_y. Z těchto dvou



Obrázek 23 – Ukázka Cannyho hranového detektoru (zdroj: [16])

gradientů lze vypočítat velikost gradientu dané vztahem 39. K dalším výpočtů potřebujeme znát také směr hrany který je dán vztahem

$$\Theta = \arctan(\frac{G_y}{G_x}). \tag{46}$$

Dalším krokem, po výpočtu hodnoty a směru gradientu, je tzv. ztenčování (*thinning*). V principu to znamená výběr lokalních maxim velikosti gradientů. Jde o postup, kdy bod je označen za hranu pouze tehdy, má-li šetřený bod větší hodnotu gradientu než dva sousední body, které jsou určeny směrem gradientu v počítaném bodě.

V předchozím kroku bylo určeno přesné umístění hran, ale nezabývalo se jejich významem. To má nastarosti poslední krok Cannyho detektoru, který se nazývá prahování s hysterezí (*thresholding*). Tuto chvíli máme označeny i ty nejmenší nevýznamné hrany, protože i ty mají své lokální maximum. Často tyto hrany můžou pocházet z šumu obrazu. Proto se užívá metoda prahování s hysterezí, která tyto nevýznamné hrany odstraní. Metoda využívá dvou hodnot prahů P_1 a P_2, kdy $P_1 < P_2$. Pokud hodnota gradientu testovaného bodu je vyšší než práh P_2, pak se bod označí jako hranový přímo. Pokud bude hodnota bodu menší než práh P_1, je označen jako nehranový. A konečně, leží-li hodnota bodu v rozmezí

hodnot prahů P_1 a P_2, pak je označen jako hranový pouze tehdy, je-li některý ze sousedních bodů již označen jako hranový. Ukázku Cannyho detektoru si můžete prohlédnout na obrázku 23.

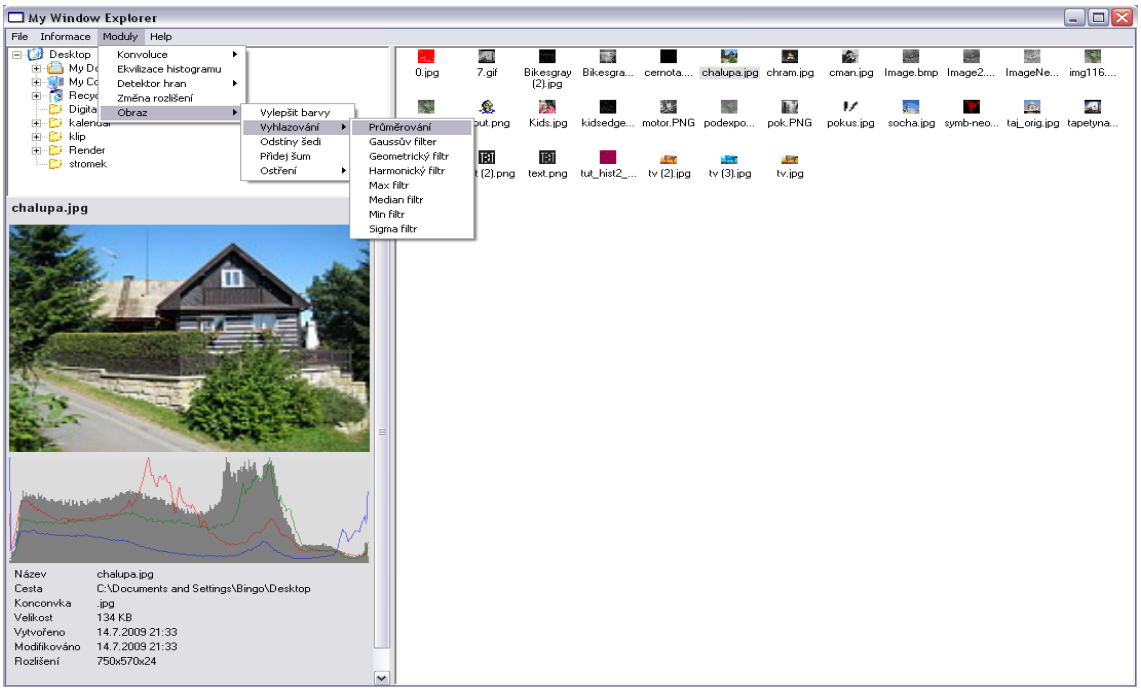
8 Implementace

Při implementaci programu výše zmíněných algoritmů jsem měl volnou ruku. Od možnosti výběru programovacího jazyka přes způsob naprogramování programu. Při výběru programovacího jazyka sem sáhl po jazyce C#, s kterým mám nejvíce zkušeností z předchozích ročníků studia. Hned od začátku byl kladen důraz na program, který bude jednoduše rozšířitelný. Důvodem byla obsáhlost oblasti předzpracování obrazu, kterou bych nebyl sám schopen obsáhnout. Rozšířitelnost programu byla docíle pomocí přidávání modulů, komunikující přes nadefinované rozhraní, o kterém se zmíním v kapitole 8.4. Díky tomuto rozhraní bylo uživatelské prostředí odděleno od logiky programu, která je prováděna právě ve vytvořených modulech.

8.1 Uživatelské rozhraní

Při spuštění programu, při kterém dochází k načítání všech modulů, se otevře okno (obr. 24), které je rozděleno na dva panely. Na levém panelu máte k dispozici průzkumníka a informace o aktuálně vybraném obrázku. V pravém panelu je zobrazen seznam obrázků v aktuálně vybraném adresáři průzkumníka. V menu programu je položka `Moduly`, která obsahuje načtené moduly nacházející ve složce `Plugins` hlavního programu. Po kliknutí na vybranou položku menu, se spustí výpočet odpovídajícího modulu následované zobrazením okna s výsledkem zpracování (obr. 25). V horní části okna je několik tlačítek, díky kterým si můžete výsledný obraz zvětšovat či zmenšovat, zobrazit v původní velikosti nebo zobrazit roztažené do velikosti okna. Pod obrázky je zobrazen uživatelský prvek, který byl vytvořen jako součást modulu. Změnou hodnoty tohoto uživatelského prvku dojde k aktualizaci změny obrazu.

Na spodní hraně je `progressBar`, který udává informaci o průběhu zpracování. Tlačítko `Uložit`, vás vyzve k zadání cesty, kam se má výsledný obraz uložit. Tlačítko `Zavřít`, uzavře formulář bez jakékoliv změny obrazu.



Obrázek 24 – Uživatelské rozhraní programu

Obrázek 25 – Okno s výsledky zpracování obrazu

8.2 Načítání obrazu

Proto aby uživatel nebyl omezen pouze na obrazy v jediném formátu, například formát BMP, umožňuje program pracovat s nejběžnějšími grafickými formáty, kterými jsou:

- BMP (Microsoft Windows Bitmap),
- JPEG (Join Photographic Experts Group),
- PNG (Portable Network Graphics),
- GIF (Graphics Interchange Format),
- TIFF (Tagged Image File Format).

Dále pak jednotlivé moduly podporují práci s 8bitovými, 24bitovými a 32bitovými obrazy.

8.3 Metoda manipulace s obrazy

Jazyk C# poskytuje několik metod pro manipulaci s bitmapou. Nejjednodušší způsobem manipulace je pomocí metod `GetPixel(x,y)` a `SetPixel(x,y, barva)` obsažených ve třídě `System.Drawing.Bitmap`. Metoda pro získání hodnoty pixelu *GetPixel(x,y)* vrací instanci struktury `System.Drawing.Color` představující barvu pixelu na pozici (x, y). Naopak metoda `SetPixel(x,y, barva)` nastavuje barvu pixelu na pozici (x, y). Následující ukázka algoritmu 1 využívá metody `GetPixel(x,y)` a `SetPixel(x,y,barva)` pro převod barevného obrazu do odstínů šedé.

Nevýhodou tohoto postupu je časová náročnost. Metody `GetPixel(x,y)` a `SetPixel(x,y,barva)` jsou výpočetně náročné operace, proto Microsoft vytvořil metodu, která pracuje přímo s pamětí obrazu. Jelikož se jedná o přistupování k paměti pomocí ukazatelů, C# nedoporučuje užívání ukazatelů, je nutné celý kód uzavřít do bloku `unsafe`. Protože pracuje přímo s pamětí, dosahuje tato technika časově neuspokojivějších výsledků při manipulaci s obrázky. Algrotimus 2 ukazuje manipulaci s obrazem v bloku `unsafe`.

Algoritmus 1 Převod barevného obrazu do odstínů šedé

```
public Bitmap PřevedNaOdstinySedi(Bitmap orig)
{
   \\ Vytovreni nove bitmapy s rozmery vstupniho obrazu
   Bitmap vysl = new Bitmap(orig.Widht, orig.Height);
   for (x = 0; x < obr.Height; x++)    // řádky
   {
      for (y = 0; y < obr.Width; y++)  // sloupce
      {
         Color barva = orig.GetPixel(x,y);
         byte seda = .299 * barva.R + .587 * barva.G + .114 * barva.B;
         vysl.SetPixel(x, y, seda);
      }
   }
   return vysl;
}
```

V definici metody je použito klíčové slovo `unsafe`, které říká, že metoda je psaná v nebezpečném módu. K tomu, aby bylo možné ve Visual Studiu kód zkompilovat, je nutné zatrhnout položku s názvem `Allow unsafe code` v menu `Project->Project Options->Debug`. Pro přístup k pixelům bitmapy je nutné bitmapu uzavřít. To se provádí pomocí metody `LockBits`, které se předá v parametru metody obdélníková oblast obrazu, která se má uzamknout. Dále pak způsob uzamčení (`ImageLockMode`), a `PixelFormat`, který určuje hloubku bitmapy.

Nastavení ukazatele na začátek bitmapy je prováděno díky vlastnosti `Scan0` třídy `BitmapData`. Procházením bitmapy pixel po pixelu, je možné aplikovat transformace hodnot jasu jednotlivých pixelů. Na konci metody musí dojít k uvolnění uzamčené oblasti bitmapy, jinak by nebylo dále možné pracovat s bitmapou.

8.4 Modulárnost programu

Modul je samostatná jednotka, která je schopná pracovat samostatně. K tomu aby mohlo jádro programu komunikovat s jednotlivými moduly, je zapotřebí na definovat komunikační rozhraní (viz. kapitola 8.4.1). Moduly se načítají při spuštění programu a nelze je přidávat ani odebírat za běhu. Aby modul byl správně načten je zapotřebí, aby splňoval následující podmínky:

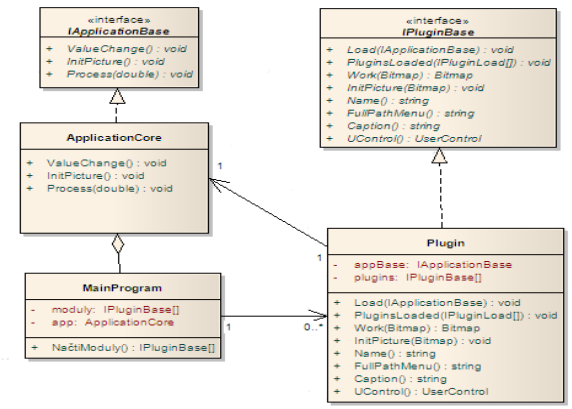
- Musí být uložen v hlavním adresáři programu ve složce nazvané `Plugin`.

Algoritmus 2 Převod barevného obrazu do odstínů šedé

```
using System.Drawing.Imaging;
using System.Drawing;

unsafe public Bitmap PřevedNaOdstinySedi(Bitmap orig)
{
   // kopie vstupního obrazu
   Bitmap vysl = new Bitmap(orig.Width, orig.Height);

   // uzamkne v paměti data vstup obrazu pouze pro čtení
   BitmapData origData = orig.LockBits(new Rectangle(0, 0, orig.Width, orig.Height),
        ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);

   // uzamkne v paměti data výstup obrazu pro zápis a čtení
   BitmapData vyslData = vysl.LockBits(new Rectangle(0, 0, vysl.Width, vysl.Height),
    ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);

   // možné rozšíření dat na každém řádku
   int offset = origData.Stride - origData.Width * 3;
   byte* pNew, pOld;
   pNew = (byte*)vyslData.Scan0.ToPointer();  // nastavení na počátek výstupních dat
   pOld = (byte*)origData.Scan0.ToPointer();  // nastevení na počátek vstupních dat

   for (int x = 0; x < origData.Height; x++)
   {
      for (int y = 0; y < origData.Width; y++)
      {
         // výpočet jasu sedeho odstinu
         byte seda = (byte)( .299 * pOld[2] + .587 * pOld[1] + .114 * pOld[0]);
         pNew[0] = pNew[1] = pNew[2] = seda;
         pNew += 3; pOld += 3;                // posun na další pixel
      }
      pNew += offset; pOld += offset; // přeskočení rozšířených dat
   }
   // otevření dat v paměti
   orig.UnlockBits(origData);
   vysl.UnlockBits(vyslData);
   return vysl;
}
```

- Musí obsahovat třídu nazvanou `Plugin`, která implementuje komunikační rozhraní `IPluginBase` z knihovny *PluginBase*.
- Celý modul musí patřit do jmenného prostoru `PluginApp`.

Celý proces komunikace jádra a modulů je na obrázku 26. Hlavní program (třída `MainProgram`) má v sobě implementované jádro (`ApplicationCore`), které funguje jako prostředník komunikace. Při spuštění se v hlavním programu nejprve načtou všechny moduly (`Plugin`). Pro každý modul se zavolá metoda `Load`. Jako parametr metody `Load` se předá instanci jádra aplikace, kterou si uloží



Obrázek 26 – Model komunikace jádra aplikace s moduly

jako atribut `app`. Tento atribut zprostředkovává zpětnou komunikaci s jádrem. Po načteních všech modulů, se zavolá pro každý z nich metoda `PluginsLoad` s parametrem `IPluginBase[]`, která předává pole načtených modulů. Díky tomuto poli je zajištěna komunikace mezi jednotlivými moduly.

Pokud dojde v jakémkoliv modulu k nějaké změně, musí se o ní dozvědět jádro aplikace. Proto modul zavolá metodu jádra nazvanou `ValueChange`. Tato metoda jádra požádá hlavní program o aktualizaci, čímž se z hlavního programu zavolá metoda modulu `Work`, která nám vrátí zpracovaný obraz.

8.4.1 Komunikační rozhraní

Knihovna `PluginBase` obsahuje dvě komunikační rozhraní (obr. 26):

- **IPluginBase** Dovoluje jádru programu komunikovat s jednotlivými moduly.
- **IApplicationBase** Dovoluje modulům komunikovat s jádrem programu.

8.4.2 Popis rozhraní `IPluginBase`

Rozhraní obsahuje čtyři metody a čtyři vlastnosti:

- Metoda `Load`, která zajišťuje předání instance jádra aplikace modulu, který bude díky této instanci schopný komunikovat s jádrem aplikace.
- Metoda `PluginsLoaded`, která parametrem předává seznam načtených modulů, díky kterým můžou jednotlivé moduly komunikovat mezi sebou.
- Metoda `Work`, která obsahuje kód pro manipulaci s obrazem. Jako parametr jí je předán obraz, který bude upravován.
- Metoda `InitPicture`, která je schopná získat informace o obraze ještě před vykonáním úprav obrazu.
- Vlastnost `Name`, která jednoznačně identifikuje název modulu.
- Vlastnost `FullPathMenu`, která dává informaci o názvu položky v menu porgramu.
- Vlastnost `UControl`, která poskytuje instanci uživatelského prvku.
- Vlastnost `Caption`, která vrací popisek okna při spuštění zpracování.

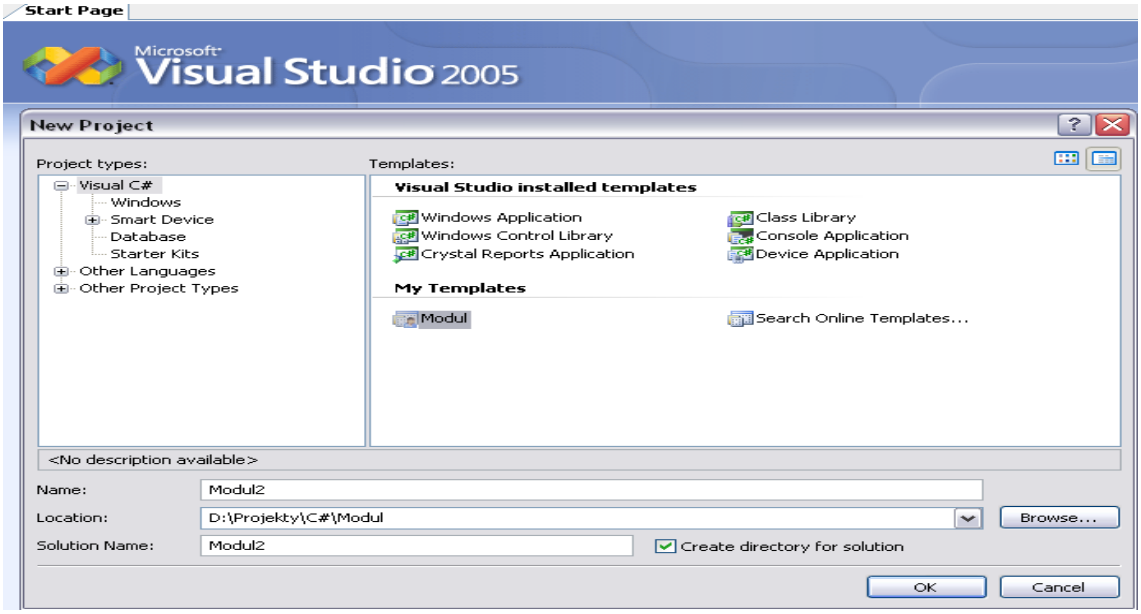
8.4.3 Popis rozhraní `IApplicationBase`

Rozhraní obsahuje tři metody:

- Metoda `ValueChange`, která informuje hlavní program o požadavku na modifikaci obrazu.
- Metoda `InitPicture`, díky které můžeme získat informace o obraze ještě před zpracováním obrazu. Provádí se pouze pro počáteční nastavení uživatelského prvku, jinak nemusí být použita.
- Metoda `Process`, aktualizuje `progress bar` hlavního okna.

8.4.4 Ukázka vytvoření modulu

Pro snadné vytváření modulů, jsem vytvořil šablonu pro Visual Studio 2005. Šablona je uložena v souboru `modul.zip`. Proto, aby bylo možné tuto šablonu využívat, musí se tento soubor zkopírovat do složky `My Documents\Visual Studio 2005\Templates\ProjectTemplates\`. Pak již stačí spustit Visual Studio, vytvořit nový projekt a vybrat šablonu `Modul` jak je ukázáno na obrázku 27.



Obrázek 27 – Výběr šablony na vytovoření modulu

Po otevření projektu je celý modul předepsán i s referencemi na potřebné knihovny. Jediné co je třeba nastavit, tak jsou vlastnosti *Nazev*, `PlnaCestaMenu`, `Titulek` a k tomu do metody `Work` vložit kód pro úpravu obrazu (můžete použít kód v algoritmu 2).

Výslednou zkompilovanou knihovnu stačí nahrát do složky `Plugins` hlavního programu. Po spuštění programu, bude váš nový modul plně připraven k užití.

9 Implementace metod a jejich výsledky

V této kapitole jsou ukázány jednotlivé implementované metody a jejich výsledky zpracování. Některé výstupní obrazy jsou porovnány s odpovídajícími metodami programu *Matlab*, který je považován za nejpropracovanější program na zpracování obrazu.

9.1 Výsledky metod jasové transformace

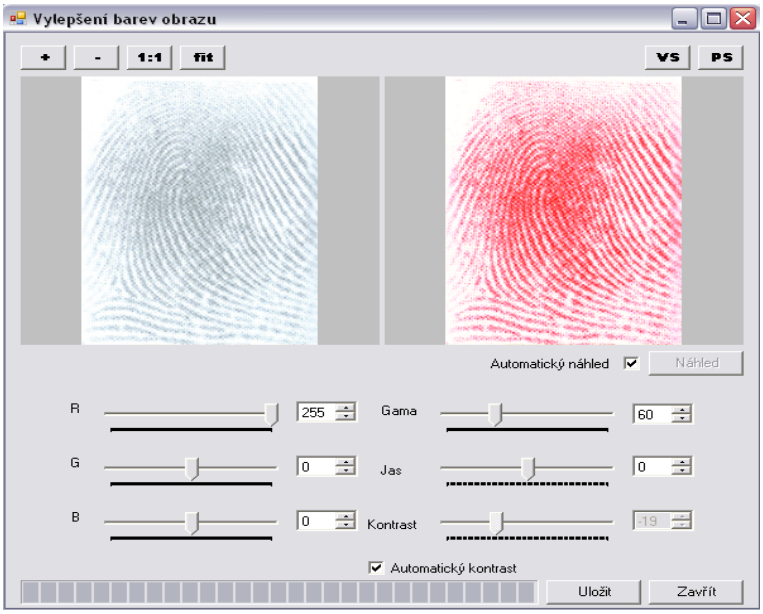
Z jasových transformací jsem implementoval pouze základní z nich. Jako například:

- Úprava barev jednotlivých kanálů R,G,B,
- úprava jasu a kontrastu,
- gama korekce,
- roztažení kontrastu.

Výsledek implementace je ukázán na obrázku 28. Na obrázku bylo nastaveno přidání červené barvy, gama korekce s koeficientem $\gamma = 0,6$ a roztažení kontrastu pro zvýraznění barev.

Pomocí posuvní, pro úpravu barevných kanálu, je možné nastavit přidávání nebo odebírání jednotlivých barev do obrazu. Rozsah hodnot je omezen na interval $< -255; 255 >$, který odpovídá 8bitům na jeden kanál. Posuvníky na přidávání, resp. odebírání, jasu a kontrastu jsou omezeny do intervalu $< -50; 50 >$, který považuji za dostačující. Posuvník pro gama korekci má interval $< 0; 200 >$, kde hodnota 0 odpovídá koeficientu $\gamma = 0$, hodnota 200 odpovídá $\gamma = 4$ a hodnota 100 odpovídá $\gamma = 1$.

Kdyby byli potřeba aplikovat větší změny jasu nebo kontrastu, je možné provést úpravy opakovaně, tj. uložení transformovaného obrazu, opětovného načtení a aplikování další transformace. Ukázka roztažení kontrastu a porovnání výsledku s výstupem z programu *Matlab*, je zobrazen na obrázku 29.



Obrázek 28 – Okno pro úpravu obrazu pomocí jasových transformací

Obrázek 29 – Ukázka porovnání metody roztažení kontrastu a) vstupní obraz, b) obraz s roztaženým kontrastem, c) obraz s roztaženým kontrastem z Matlabu

Na první pohled je patrné, že program *Matlab* roztáhl kontrast o trochu více než má metoda. Zřejmě používá lepší metody pro výpočet intervalu hodnot jasu, do kterého lze obraz roztáhnout. Script na roztažení kontrastu v programu *Matlab* je ukázán v algoritmu 3.

Algoritmus 3 Funkce napsaná v *Matlabu* pro roztažení kontrastu obrazu

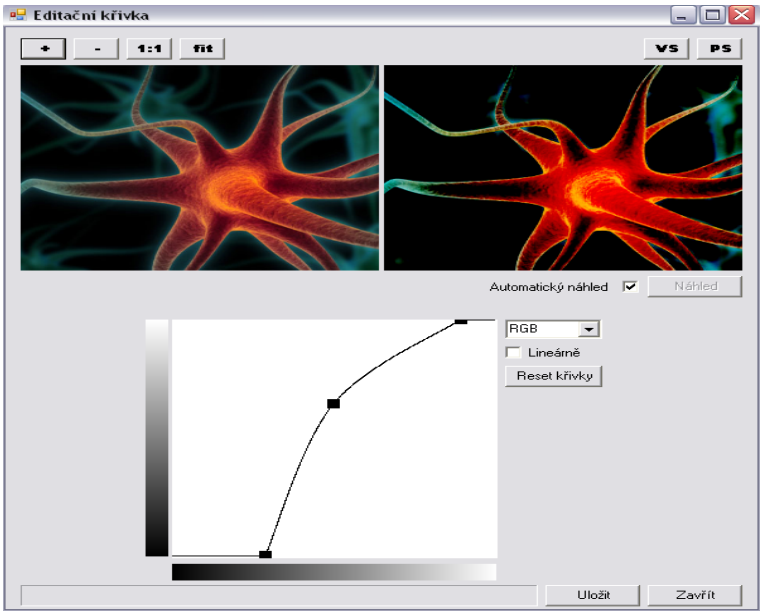
```
function ContrastStretch(cesta)
   % parametrem metody je cesta k nazvu souboru
   vstup = imread(cesta); % nacteni obrazu
   imshow(vstup); % zobrazeni vstupniho obrazu
   % aplikace roztazeni
   vystup = imadjust(vstup,stretchlim(vstup),[]);
   figure; % vytvoreni noveho okna pro vystupni obraz
   imshow(vystup); % zobrazeni obrazu do noveho okna
```

9.2 Výsledky metod založené na změnách histogramu

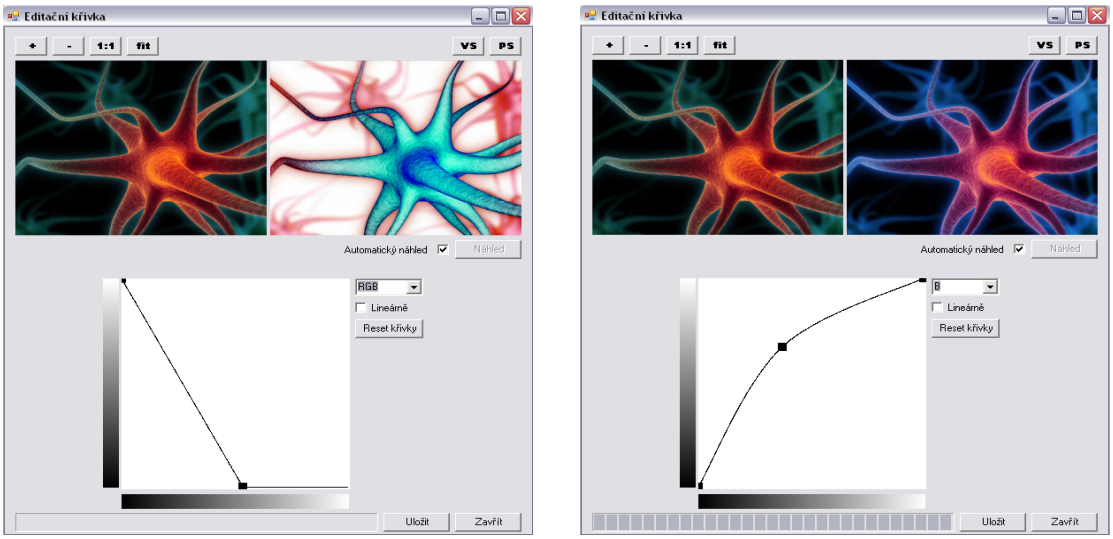
Transformace založené na změnách histogramu jsou implementovány pomocí editační křivky s kterou je možné provádět operace změny hodnot jasu, kontrastu, převod obrazu na negativ, změny bílého a černého bodu a také prahování. Okno s implementovanou křivkou je na obrázku 30.

V poli křivky lze pohybovat jednotlivými body do požadované polohy pro úpravu obrazu. Lze také přidávat jednotlivé body na křivku, čímž lze upřesnit průběh křivky. Volbou kanálu lze editovat všechny kanály najednou nebo pouze vybraný z nich. Dále je možné volit mezi kreslením křivky mezi jednotlivými body nebo kreslit lineární čáry. Pomocí tlačítka `Nuluj` vrátíte posunuté body zpět do počáteční polohy. Na obrázku 31 jsou ukázky některých výsledných obrazů i s odpovídajícími křivkami.

Algoritmus pro zvýšení výkonu používá vyhledávací tabulku, která je vypočítána pouze pro 255 odstínu jasu a poté přiřazena odpovídající hodnotě jasu. Díky tomuto navýšení rychlosti je algoritmus schopen pracovat v reálném čase.



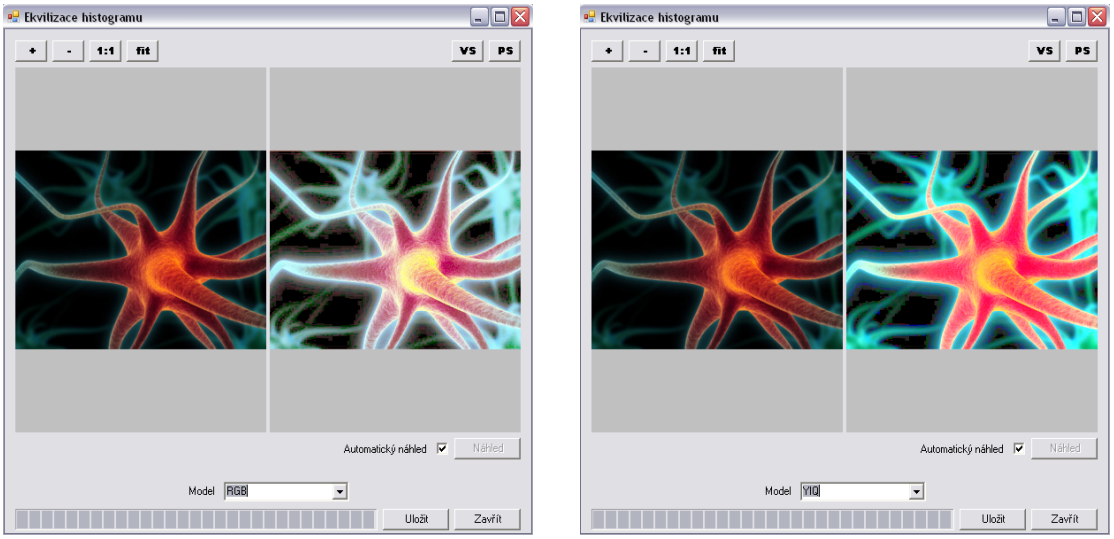
Obrázek 30 – Okno pro úpravu obrazu pomocí editační křivky



Obrázek 31 – Ukázky výsledků editační křivky a) negativ s posunem bílého bodu b) zvýšení jasu modrého kanálu

9.2.1 Ekvilizace histogramu

Ekvilizace histogramu je operace změny histogramu, která roztáhne hodnoty jasu rovnoměrně po celém rozsahu hodnot jasu. Okno s výsledkem implementované metody je na obrázku 32. V programu je možné volit mezi aplikací v modelu barev *YIQ* nebo v modelu *RGB*. Ukázky výsledků obou modelů je na obrázku 32.



Obrázek 32 – Okna pro ekvilizaci histogramu a) model RGB, b) model YIQ

Program *Matlab* poskytuje funkci `histeq(obraz)`, která umí ekvilizovat pouze 8bitové obrazy v odstínech šedi. Vytvovřená funkce, v programu *Matlab*, pro ekvilizaci histogramu je ukázán v algoritmu 4. Výsledky metod lze porovnat na obrázku 33.

Algoritmus 4 Funkce napsaná v *Matlabu* pro ekvilizaci histogramu

```
function EkvilizaceHistogramu(cesta)
   % parametrem metody je cesta k nazvu souboru
   vstup = imread(cesta); % nacteni vstupniho obrazu
   imshow(vstup); % zobrazeni vstupniho obrazu
   vystup = histeq(vstup); % aplikace metody ekvilizace histogramu
   figure; imshow(vystup); % zobrazeni vystupniho obrazu
```

Obrázek 33 – Porovnnání výsledků metod ekvilizace histogramu a) vstupní obraz, b) ekvilizovaný obraz, c) ekvilizovaný obraz z *Matlabu*

Z ukázky výsledků se dá říct, že výsledek, mé implementované metody a metody z programu *Matlab*, je identický. Žádné rozdíly v hodnotách jasu, ani rozdíly v zobrazení hran nejsou rozpoznatelné.

9.3 Výsledky interpolace jasu v obraze

Metody interpolace jasu jsou implementovány pro změny měřítka obrazu. Výběrem metody interpolace a nastavení výsledné šířky a výšky obrazu získáte výsledný obraz. Šířku a výšku obrazu lze nastavovat jak v hodnotách pixelů, tak i procentuálně. Interpolační metody lze vybírat v okně `Metody`. Na výběr je hned několik metod, kterými jsou:

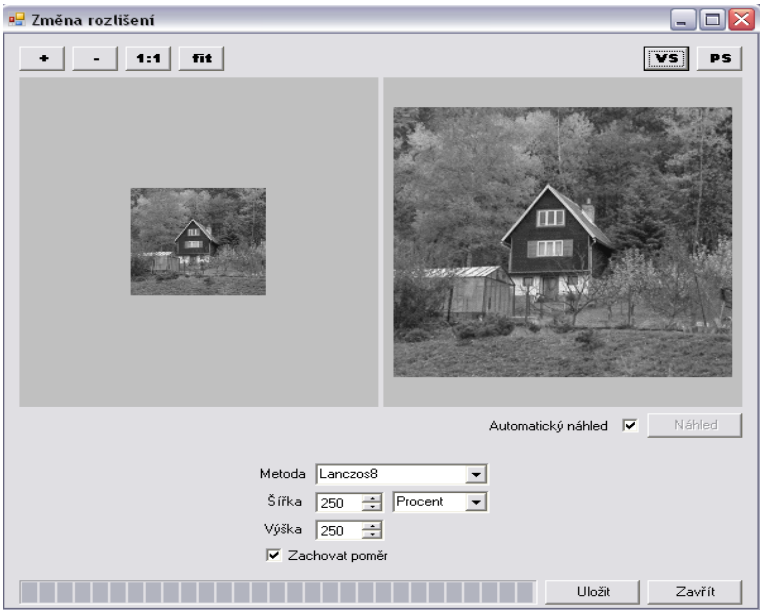
- interpolace nejbližším sousedem (*box kernel*),
- bilineární interpolace (*triangle kernel*),
- bikubická interpolace (*cubic kernel*),
- spline interpolace (*cubic spline kernel*),
- interpolace Lanczosovým oknem (*Lanczos3*, *Lanczos8*).

Jednotlivé metody jsou implementovány pomocí konvolučních masek (kernel), složitost dvou rozměrné konvoluce je $O(width \times height \times raius^2)$, kde $radius$ je rozměr masky. Díky vlastnosti separability těchto masek, dochází k navýšení

Algoritmus 5 Funkce napsaná v *Matlabu* pro změnu rozlišení obrazu

```
function Interpolace(cesta, meritko, metoda)
   % cesta -> cesta k vstupnimu obrazu
   % meritko -> hotnota m = <0,1> pro zmenseni obrazu
   %         -> hodnota m > 1 pro zvetseni obrazu
   % metoda -> nazev metody interpolace
   %    'nearest', 'bilinear', 'bicubic',
   %    'box', 'triangle', 'cubic', 'lanczos3'

   vstup = imread(cesta); % nacteni vstupniho obrazu
   imshow(vstup); % zobrazeni vstupniho obrazu
   vystup = imresize(vstup, meritko, metoda); % zmena meritka zadanou metodou
   figure, imshow(vystup); % zobrazeni vystupniho obrazu
```

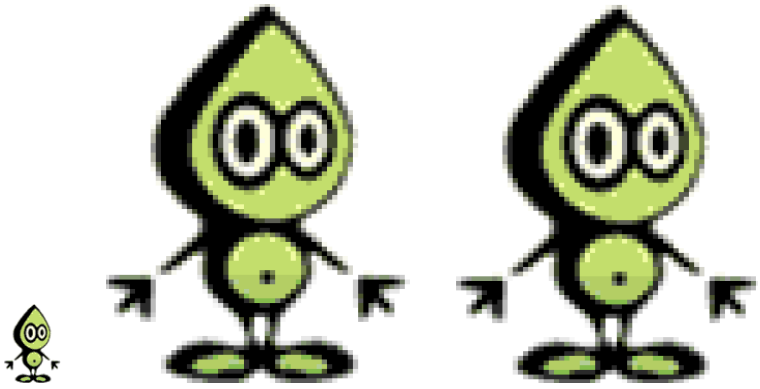
výkonu celého algoritmu. Složitost separabilní interpolace je $O(width \times height \times radius)$. Výsledek implementace interpolace jasu je na obrázku 34.



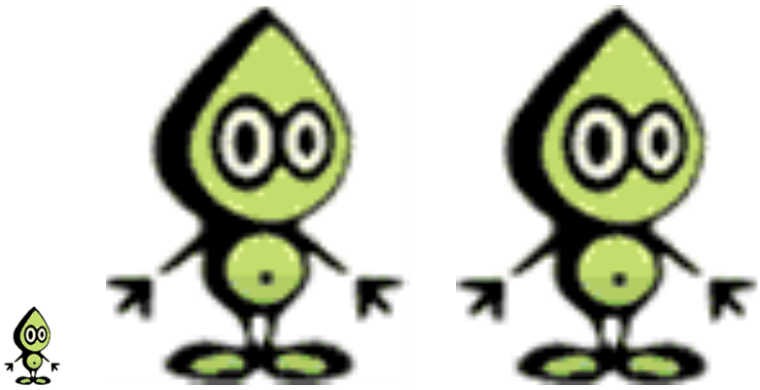
Obrázek 34 – Zvětšení obrazu (2,5 krát) metodou Lanczos8

Software *Matlab* nabízí v balíku *Image processing* metodu pro změnu měřítka obrazu nazvanou `imResize`. V parametrech metody se předává načtený vstupní obraz, hodnotu měřítka změny obrazu ($m \in < 0; 1 >$ pro zmenšení, $m > 1$ pro zvětšení) a název použité interpolační metody. Funkce pro změnu rozlišení v pro-

gramu je k vidění v algoritmu 5. Porovnání výsledků implementovaných metod s výstupy z programu *Matlab* jsou zobrazeny na obrázcích 35, 36, 37 a 38.



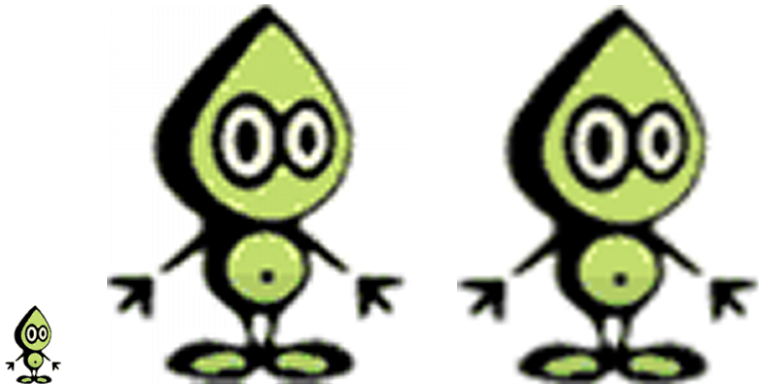
Obrázek 35 – Metoda nejbližší soused a) vstupní obraz, b) nejbližší soused, c) nejbližší soused z *Matlabu*



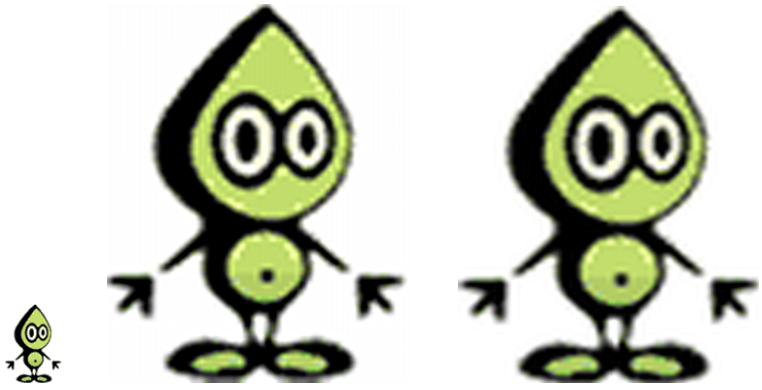
Obrázek 36 – Metoda bilinearni interpolace a) vstupní obraz, b) bilineární interpolace, c) bilineární interpolace z *Matlabu*

Implementované metody byli testováný jak na zvětšení obrazu(viz obr. 39), tak i na jeho zmenšení (viz obr. 40). Měřítko m pro zvětšení obrazu bylo rovno $m = 8$. Pro zmenšování obrazu bylo měřítko nastaveno na hodnotu $m = 0,3$.

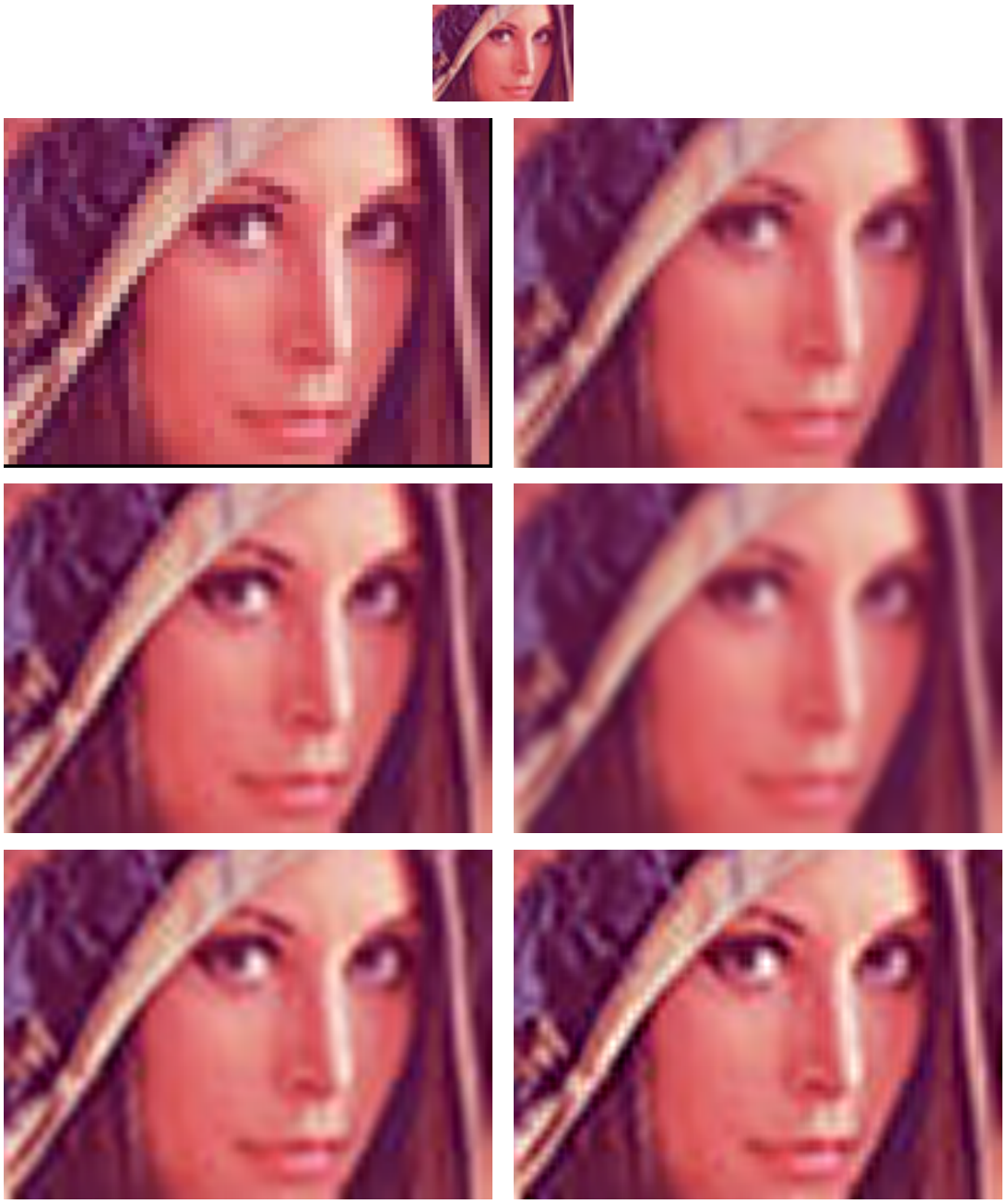
Metoda interpolace nejbližším sousedem ukazuje nejhorší výsledky změny velikosti ze všech. Nepoužívá žádnou interpolaci, pouze přiřazuje nejbližší bod vstupního obrazu. Proto tato metoda není vůbec vhodná pro obrazy typu foto-



Obrázek 37 – Metoda bikubické interpolace a) vstupní obraz, b) bikubická interpolace, c) bikubická interpolace z *Matlabu*



Obrázek 38 – Lanczosova metoda a) vstupní obraz, b) Lanczos3, c) Lanczos3 z *Matlabu*



Obrázek 39 – Výsledky interpolačních metod pro $m = 8$ a) vstupní obraz, b) nejbližší soused, c) bilineární interpolace, d) bikubická interpolace, e) spline interpolace, g) Lancosz3, h) Lancosz8

Obrázek 40 – Výsledky interpolačních metod pro $m = 0,3$ a) vstupní obraz, b) nejbližší soused, c) bilineární interpolace, d) bikubická interpolace, e) spline interpolace, g) Lancosz3, h) Lancosz8

grafie. Je však neocenitelná při změně velikosti v technických kresbách vlasovými čarami.

Metoda bilineární interpolace využívá čtyř sousedních bodů, z kterých vypočítává výslednou hodnotu jasu. Obecně podává lepší výsledky než interpolace nejbližším sousedem. Je to také nejrychlejší metoda, která k výpočtu výsledného bodu, užívá interpolaci hodnot jasu okolních pixelů. Na výsledku zvětšení ukazuje příliš vyhlazené hrany v obraze. Proto není vhodná pro zvětšování obrazu. Při zmenšování obrazu dává celkem uspokojivé výsledky. Proto se svoji rychlostí se doporučuje pro vytváření kvalitnějších náhledů na obrazy.

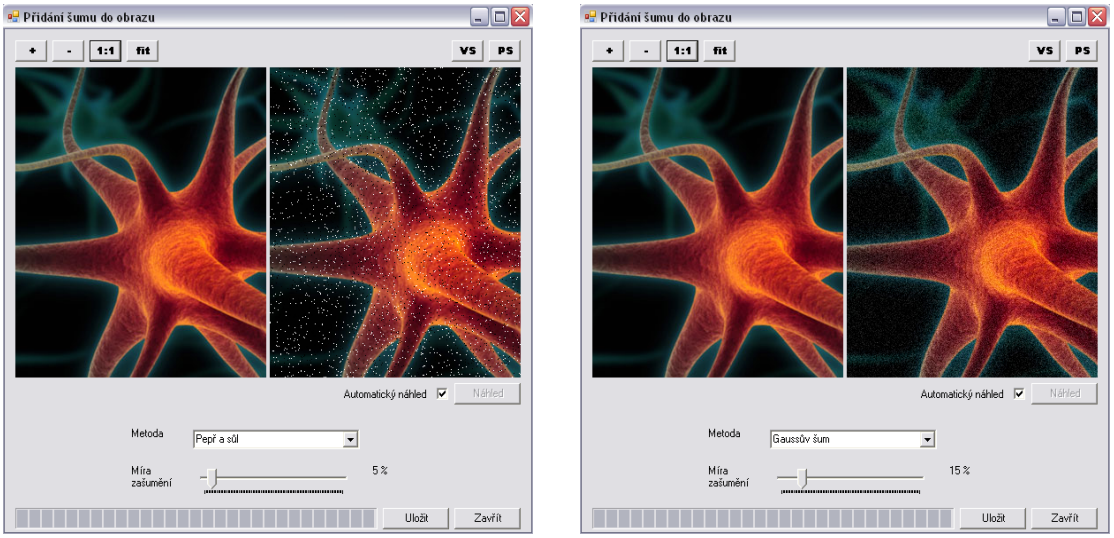
Metoda bikubické interpolace používá, k výpočtu výsledné hodnoty jasu, šestnáct sousedících pixelů. To má za následek pomalejší zpracování než předchozí metody. Metoda prokládá těchto šestnáct pixelů křivkou, díky které je hodnota jasu přesnější. Z výsledků změny měřítka obrazu vyplývá, že metoda je vhodná jak ke zvětšování tak i zmenšování obrazu. V kombinaci s ostřícími metodami mohou být dosaženy výborné výsledky změny rozlišení.

Podobně, jako metoda bikubické interpolace, je na tom bikubická interpolace pomocí splain křivek. V některých případech dokáže podat lepší výsledky než metoda bikubické interpolace. V našem případě však velmi vyhladil hrany obrazu. Opět pomocí ostřících metod lze podat daleko lepších výsledků.

Lanczosovy okna jsou pro výpočet hodnot výsledných jasů považovány za nejlepší metody při zvětšování obrazu. Za kvalitní výsledky však platí časovou náročností výpočtu, protože okna použita při výpočtu jsou příliš velká. Proto se tato metoda určitě nevyužije v systémech, na které jsou kladeny požadavky rychlého zpracování. Při snižování velikosti mohou, kvůli velikosti masky, opomenout některé detaily v obraze. Dále z výsledků je patrné, že mají samozaostřovací účinek, proto již není nutné aplikovat ostřící metodu pro zkvalitnění výsledku.

9.4 Výsledky filtračních metod

K tomu aby mohli být testovány filtrační metody je zapotřebí získat zašuměné obrazy. Pro tento účel byl vytvořen modul, který aplikuje umělé zašumění na obraz typu *Sůl a pepř*, pouze *Sůl*, pouze *Pepř* a *Gaussův šum*. Okna výsledného



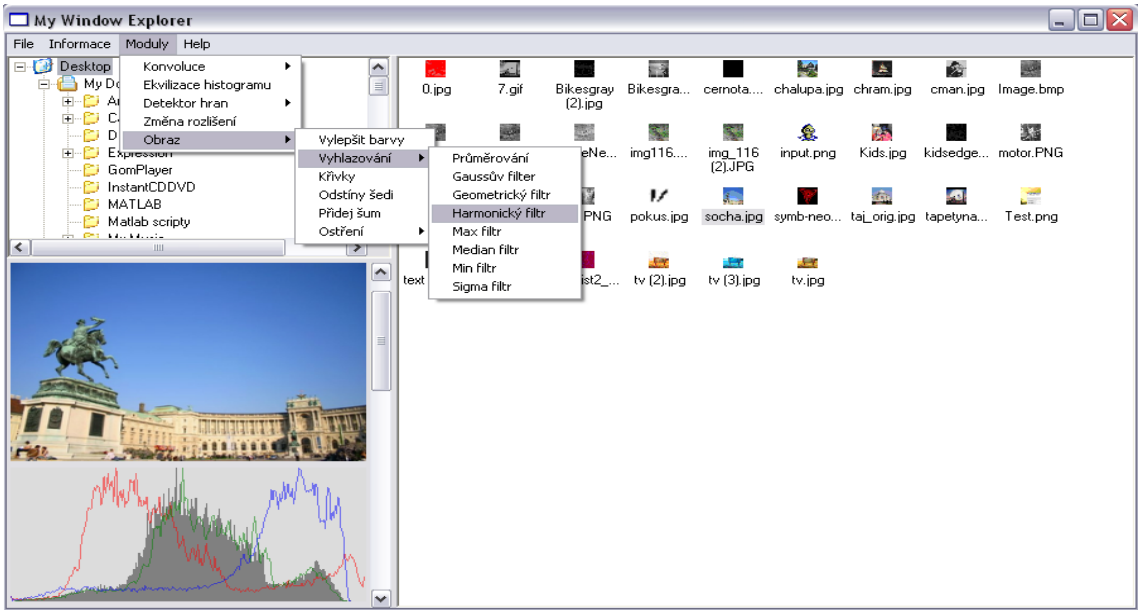
Obrázek 41 – Modul pro přídání umělého šumu a) šum *Sůl a pepř* b) *Gaussův šum*

modulu pro přidání šumu typu *Sůl a pepř* a *Gaussova šumu* jsou ukázána na obrázku 41. Volbou metody zašumění a nastavením míry zašumění je možné měnit výsledný vzhled obrazu.

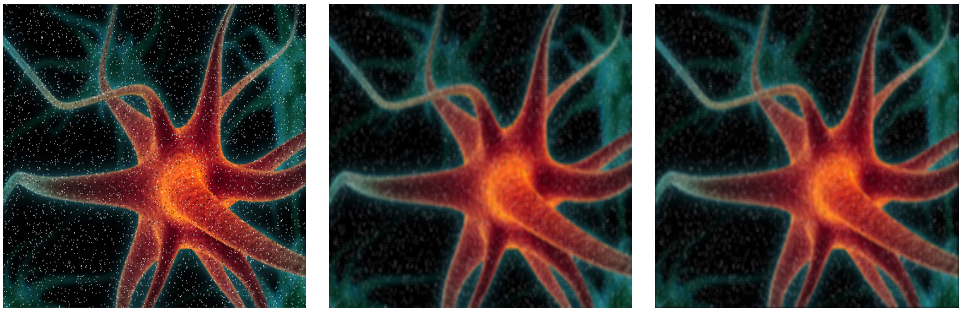
Když získáme zašuměné obrazy je vhodné tyto vady z obrazu odstranit. K tomu nám slouží metody filtrace. Každá implementovaná metoda je specializována na různé typy zašumění, proto správný výběr metody je velmi důležitý. Na obrázku 42 je ukázán výčet implementovaných metod v programu.

Program *Matlab* má k dispozici v balíku *Image Processing* několik metod na odstraňování šumu. Některé základní budou ukázány na porovnání výsledků. Na obrázku 43 je porovnání výsledků metod průměrování. Funkce napsaná v programu *Matlab* je ukázána v algoritmu 6.

Další metoda, která byla porovnávána je metoda vyhlazování obrazu nazvaná *Gaussovo vyhlazování*. Výsledek si můžete prohlédnout na obrázku 44. Při testování byla velikost masky nastavena na $[7, 7]$ a směrodatná odchylka $sigma = 1$. V algoritmu 7 je ukázána funkce napsaná pro testování metody *Gaussova vyhlazování*.



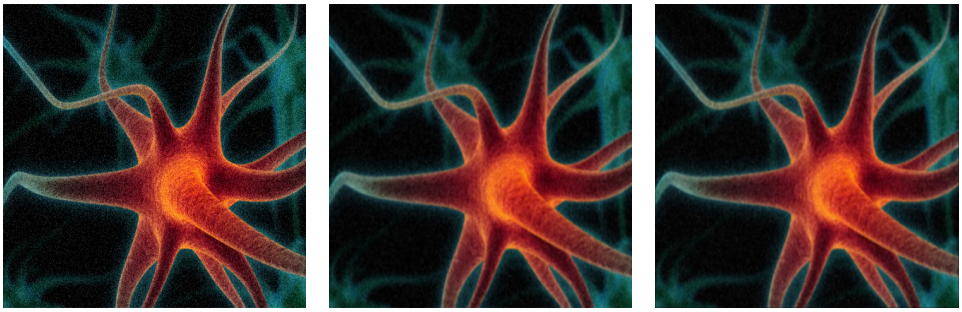
Obrázek 42 – Přehled implementovaných metod filtrace



Obrázek 43 – Porovnání metody průměrování s velikostí masky $[5, 5]$ a) vstupní obraz b) metoda průměrování c) metoda průměrování z *Matlabu*

Algoritmus 6 Funkce v *Matlabu* pro vyhlazování obrazu metodou průměrování

```
function Prumerovani(cesta, velikostMasky)
   % parametr cesta -> urcuje cestu k souboru
   % parametr velikostMasky -> urcuje velikost masky prumerovani
   % defaultne velikost maksy [3 3]
   %     [5 5], [7 7], ...

   % test na nastaveni velikosti masky filtru
   if exist(velikostMasky) == 0
      velikostMasky = [3 3];
   end

   vstup = imread(cesta); % nacteni vstupniho obrazu
   figure, imshow(vstup); % zobrazeni vstupniho obrazu
   filtr = fspecial('average', velikostMasky); % vytvoreni filtru
   vystup = imfilter(vstup, filtr); % aplikace filtru
   figure, imshow(vystup); % zobrazeni vystupniho obrazu
```



Obrázek 44 – Porovnání metody Gaussova vyhlazování s velikostí masky $[7, 7]$ a $sigma = 1$ a) vstupní obraz b) Gaussovo výhlazování c) Gaussovo vyhlazování z *Matlabu*

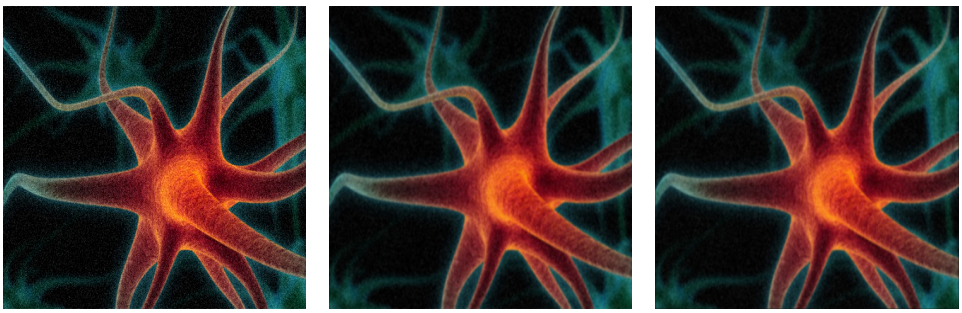
Poslední porovnávaná metoda byla metoda mediánové filtrace. Výsledky metod jsou zobrazeny na obrázku 45. Funkce napsaná pro podporu testování je ukázána v algoritmu 8.

Bohužel, mnou implementovaná metoda mediánové filtrace nepodala tak dobré výsledky jako metoda `medfilt2` z programu *Matlab*. U testů s velikostí masky $[3, 3]$ zůstaly některé pixely neopraveny. U testů s velikostí masky $[5, 5]$ jsou už výsledky shodné. U předchozích dvou metod se dají výsledky metod prohlásit shodné.

Testování odstraňování šumu bylo aplikováno na dva obrazy. Jeden obraz se šumem typu *sůl a pepř* (viz obr. 46) a druhý obraz s *Gaussovým šumem* (viz obr. 47).

Algoritmus 7 Funkce v *Matlabu* pro vyhlazování obrazu Gaussovým filtrem

```
function GaussuvFiltr(cesta, velikostMasky, smerOdchylka)
   % parametr cesta -> urcuje cestu k souboru
   % parametr velikostMasky -> urcuje velikost masky prumerovani
   % defaultne velikost maksy [3 3]
   %     [5 5], [7 7], ...
   % parametr sigma udava

   % test na nastaveni velikosti masky filtru
   if (isempty(velikostMasky))
      velikostMasky = [3 3];
   end

   vstup = imread(cesta); % nacteni vstupniho obrazu
   figure, imshow(vstup); % zobrazeni vstupniho obrazu
   % vytvoreni filtru gaussian
   filtr = fspecial('gaussian', velikostMasky, smerOdchylka);
   vystup = imfilter(vstup, filtr); % aplikace filtru
   figure, imshow(vystup); % zobrazeni vystupniho obrazu
```

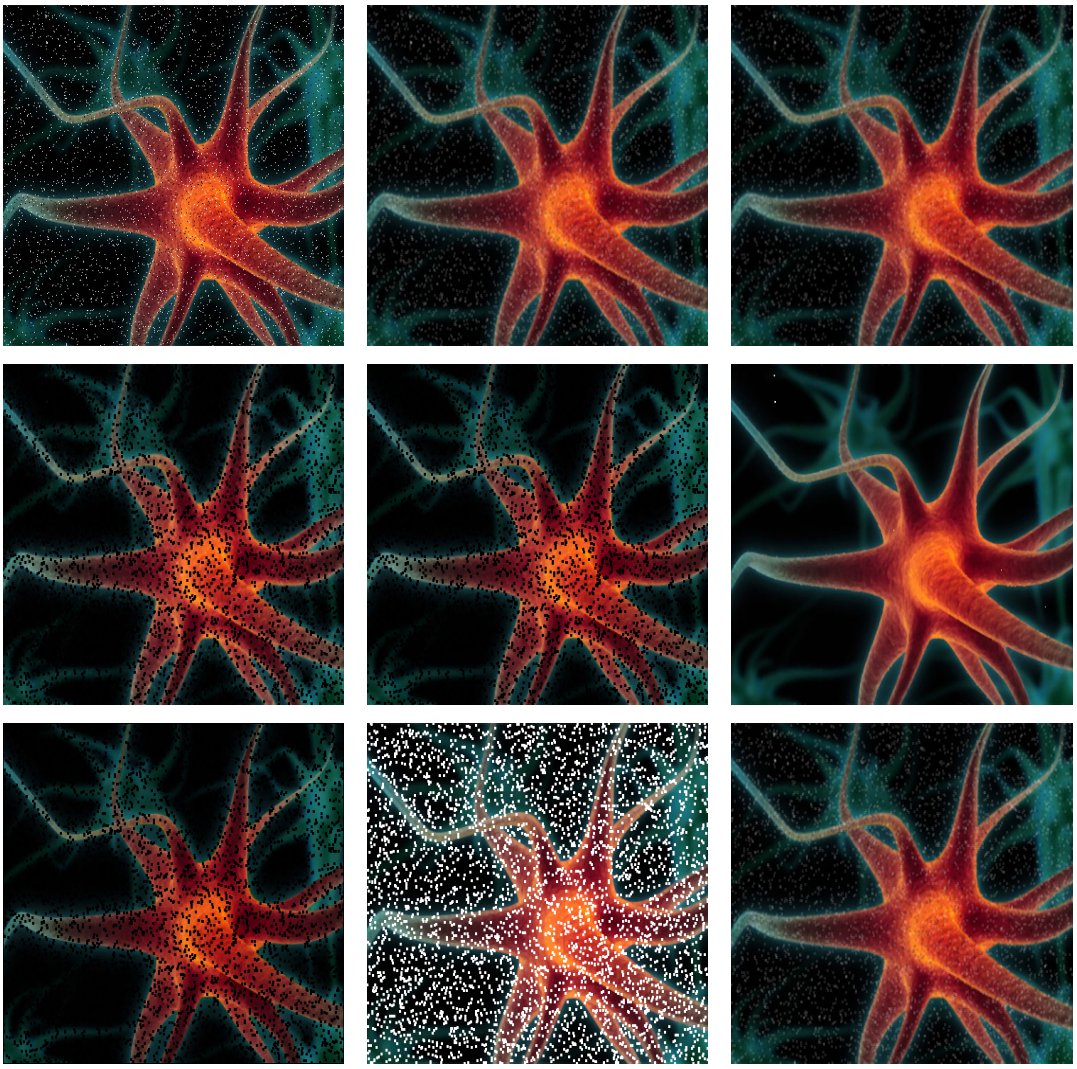


Obrázek 45 – Porovnání mediánové filtrace s velikostí masky $[5, 5]$ a) vstupní obraz b) Mediánový fitr, c) Mediánový filtr z *Matlabu*

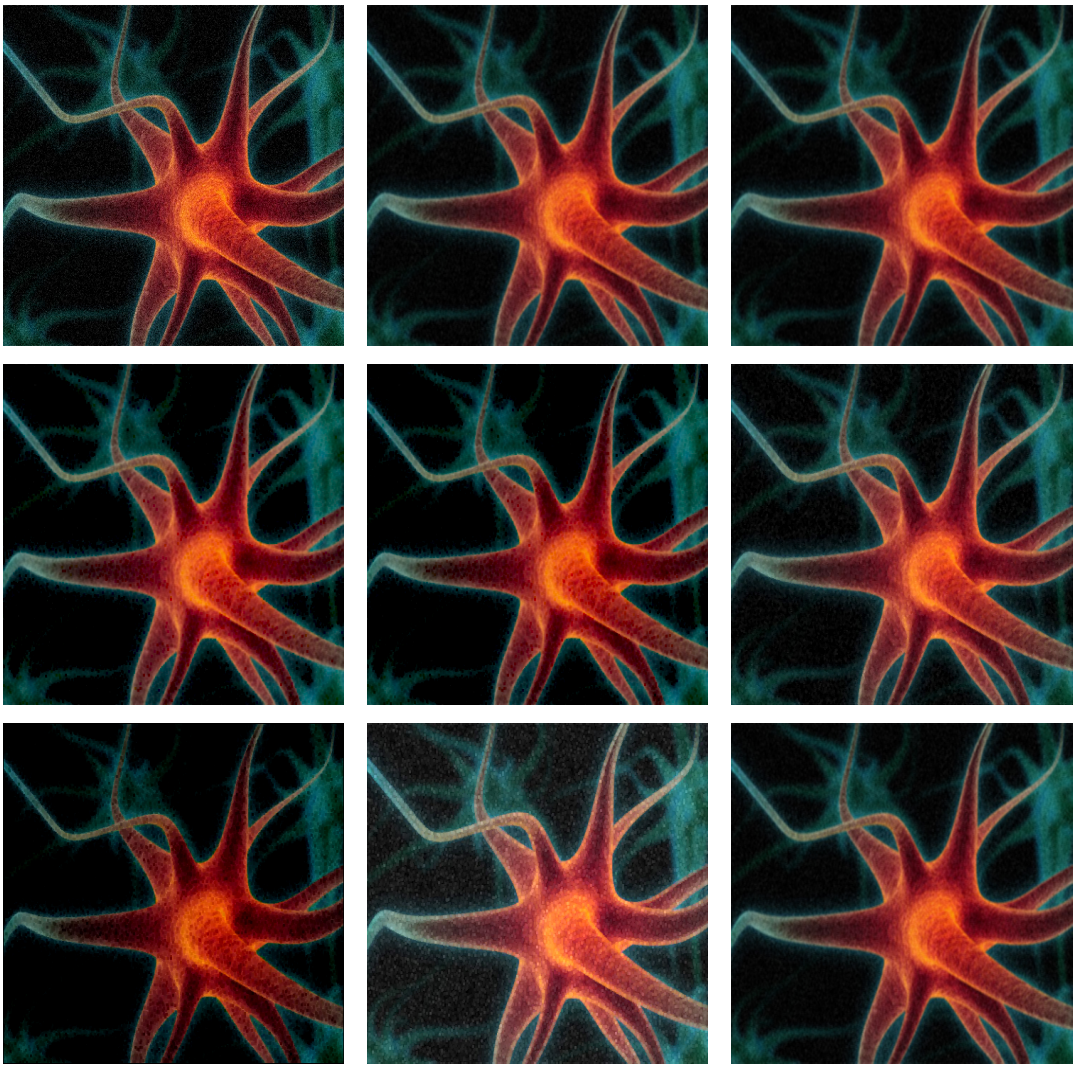
Podle očekávání metoda vyhlazování průměrováním nepodala nijak oslnivé výsledky. Při odstraňování obou typů šumu podal podprůměrné výsledky. Celkově tedy není vhodný pro odstraňování šumu z obrazu. Jedním místem jeho využití je, pokud vznikne potřeba obraz rozmazat.

Gaussův filtr není na tom o moc víc lépe než jeho předchůdce. U obou typů šumu podal porovnatelné výsledky, u odstraňování šumu typu *Gaussův šum*, podal lepší výsledky než ostatní metody. Je pouze nutné najít správné nastavení směrodatné odchylky. Potom je schopný podat dobré výsledky.

Geometrický filtr se zaměřuje šum typu *sůl* (viz obr. 48). Při odstraňování tohoto šumu podává mnohem lepší výsledky než metoda průměrování. Na šum



Obrázek 46 – Výsledky odstraňování šumu typu *sůl a pepř* s velikostí masky $[3, 3]$
a) vstupní obraz, b) metoda průměrování c) Gaussovo vyhlazování, d) Geometrický filtr, e) Harmonický filtr, f) filtrace mediánem, g) Min filtr, h) Max filtr i) Sigma filtr



Obrázek 47 – Výsledky odstraňování Gaussového šumu s velikostí masky $[3, 3]$
a) vstupní obraz, b) metoda průměrování c) Gaussovo vyhlazování, d) Geometrický filtr, e) Harmonický filtr, f) filtrace mediánem, g) Min filtr, h) Max filtr i) Sigma filtr

Algoritmus 8 Funkce v *Matlabu* pro vyhlazování obrazu metodou mediánové filtrace

```
function MedianFiltr(cesta, velikostMasky)
   % parametr cesta -> urcuje cestu k souboru
   % parametr velikostMasky -> velikost masky prumerovani
   % defaultne velikost maksy [3 3]
   %     [5 5], [7 7], ...

   % test na nastaveni velikosti masky filtru
   if (isempty(velikostMasky))
      velikostMasky = [3 3]; % defaulni hodnota
   end

   vstup = imread(cesta); % nacteni vstupniho obrazu
   figure, imshow(vstup); % zobrazeni vstupniho obrazu
   vystup = medfilt2(vstup, velikostMasky); % aplikace
   figure, imshow(vystup); % zobrazeni vystupniho obrazu
```

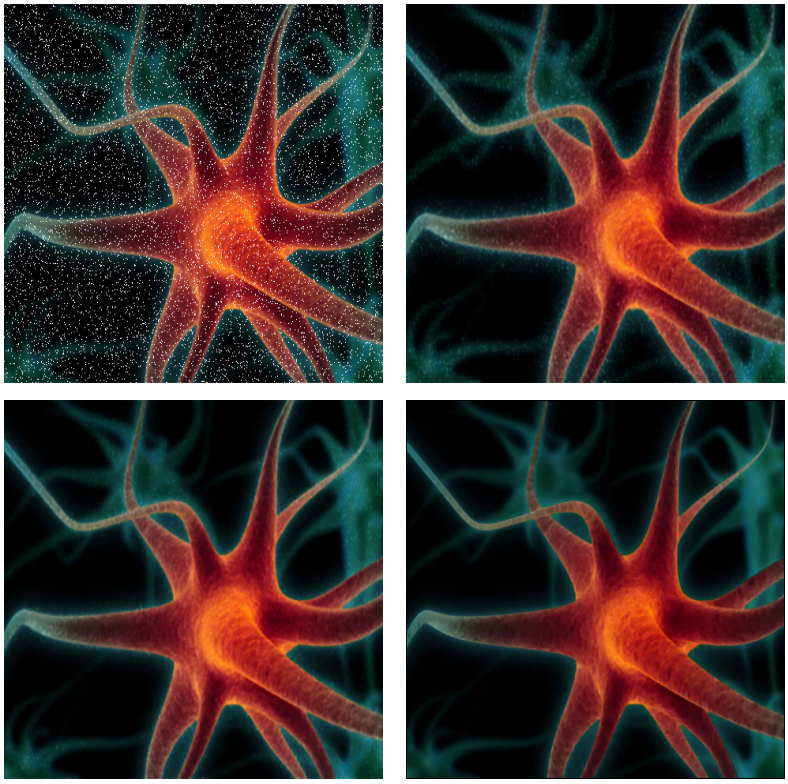
typu *pepř* nereaguje dobře. Porušený obraz černými pixely jsou ve výsledném obrazu ještě zvětšeny. Hrany v obraze nejsou zachovány. Proto se užívá jen v některých případech.

Harmonický filtr je na tom podobně jako Geometrický filtr. Také se zaměřuje na šum typu *sůl* (viz obr. 48). Podává však lepší výsledky. Hlavní výhodou je, že zachovává hrany obrazu.

Mediánový filtr podal nejlepší výsledky u šumu typu *sůl a pepř*. Obecně je to jeden z nejpoužívanější filtrů na odstraňování šumu. Proto by neměl chybět v žádném programu odstraňující šum.

Min a Max filtry jsou zvláštní případy filtrování mediánem. Min filtr, resp. Max filtr, vybírá první, resp. poslední, hodnotu uspořádané posloupnosti hodnot jasů okolních pixelů. Proto se Min filtr specializuje na šum typu *sůl* (viz obr. 48) a Max filtr na šum typu *pepř*.

Sigma filtr podává o něco lepší výsledky než vyhlazování průměrováním u šumu typu *sůl*. Využívá toho, že se snaží eliminovat body, které vypadají jako šum. Proto pokud je metoda implementována, stojí za to zvážit, zda nedat přednost této metodě před vyhlazování průměrováním.



Obrázek 48 – Odstranění šumu typu *sůl* a) zašuměný obraz, b) Geometrický filtr, c) Harmonický filtr, d) Min filtr

9.5 Výsledky detektorů hran

V programu bylo implementováno několik durhů detektorů hran, kterými jsou:

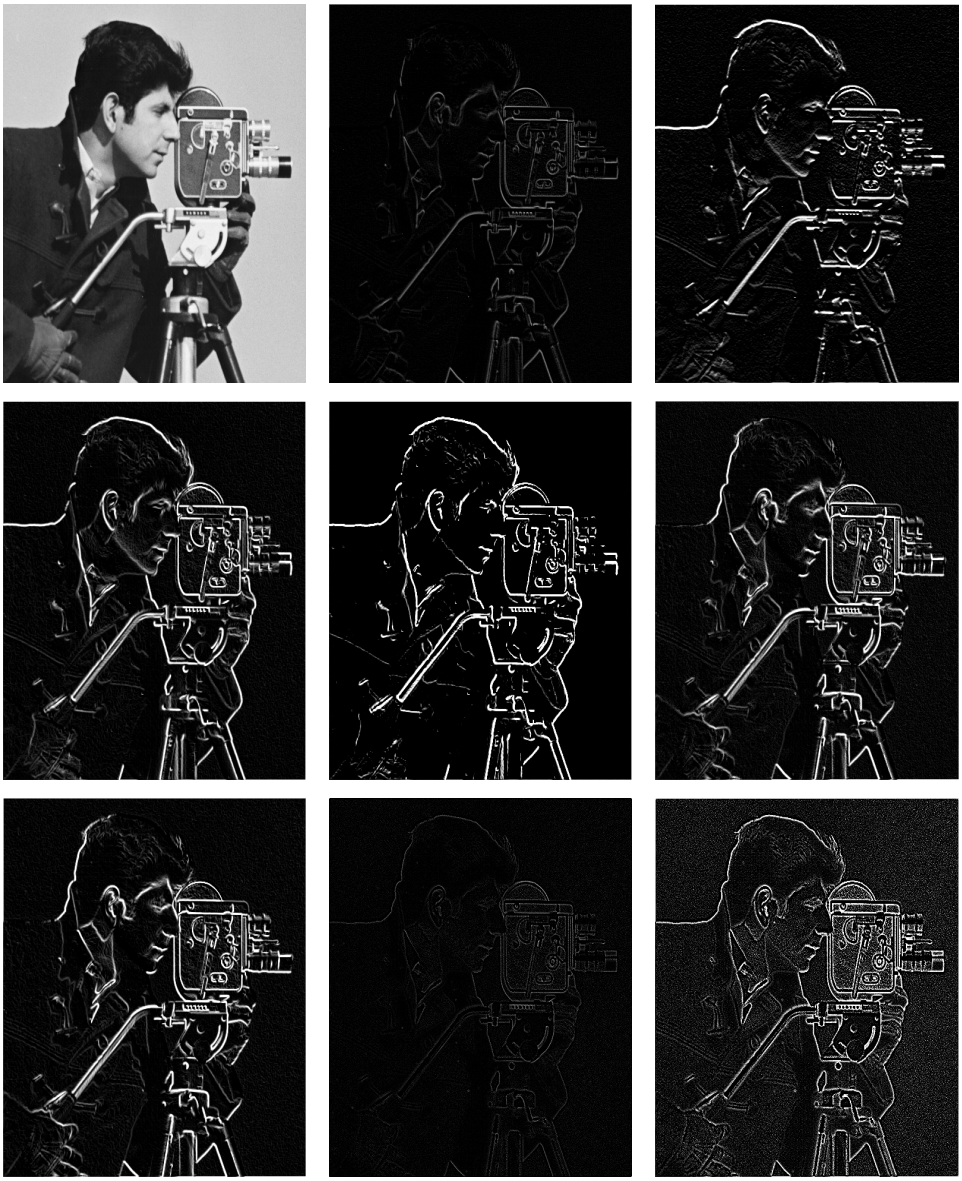
- Detektory založené na výpočtu první derivace (Roberts, Robinson, Kirchs, Prewit, Sobel, Frei Chen) implementováné pomocí konvoluce.
- Detektory založené na výpočtu druhé derivace (Laplacův operátor, Laplacian of Gaussian) implementované pomocí konvoluce.
- Detektor Laplacian of Gaussian (LoG), diference Gaussiánu (DoG).
- Cannyho hranový detektor.

Obrázek 49 – Výsledné okno detekce hran pomocí výpočtu konvoluce

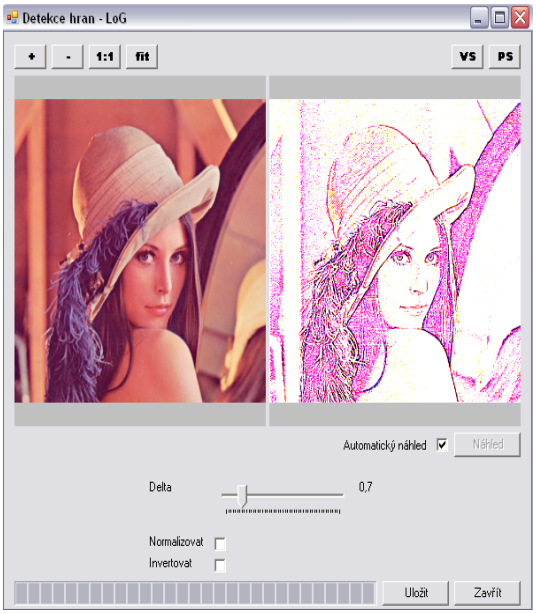
Metody detekce hran pomocí konvoluce se nacházejí v menu `Moduly -> Detekce hran -> Konvoluce`, který je ukázán na obrázku 49. Po výběru metody detekce v okně `Metody` se spustí vybraný detektor hran. Ukázky výsledků jednotlivých metod jsou zobrazeny na obrázku 50.

Z výsledků hranové detekce pomocí konvoluce plyne, že není podstatné, které metody se užijí. Výsledky metod využívající první derivaci jsou celkem shodné. Pro zlepšení výstupních obrazů je možné aplikovat metodu prahování, která odstraní nevýznamné hrany. Tím zbudou pouze významné hrany v obraze, které jsou pří hledání hran podstatné. Prahování je možné nastavit pomocí posuvníku nazvaného `Práh`. Metody druhé derivace ukazují mnohem přesnější nalezení hran. Laplacian of Gaussian ukazuje krásné nalezení hran, které nejsou rotrhané jako u předchozích metod. Pomocí prahování lze také odstranit nevýznamné hrany a tím docílit lepšího výsledku.

Metoda Laplacian of Gaussian (LoG), která nevyužívá k výpočtu konvoluci, je metoda detekce hran, která není tak citlivá na šum. Protože obraz je nejprve vyhlazen (Gaussovo vyhlazování) a poté jsou hledány hrany pomocí Laplaceova operátoru. Proto tato metoda (viz obr. 51) podává dobré výsledky. Směrodatná odchylka



Obrázek 50 – Výsledky detektorů hran počítaných pomocí konvoluce s prahem $p = 1$
a) vstupní obraz, b) Roberts, c) Sobel, d) Prewitt, e) Kirsch, f) Robinson, g) Frei Chen,
h) Laplacian, i) Laplacian of Gaussian



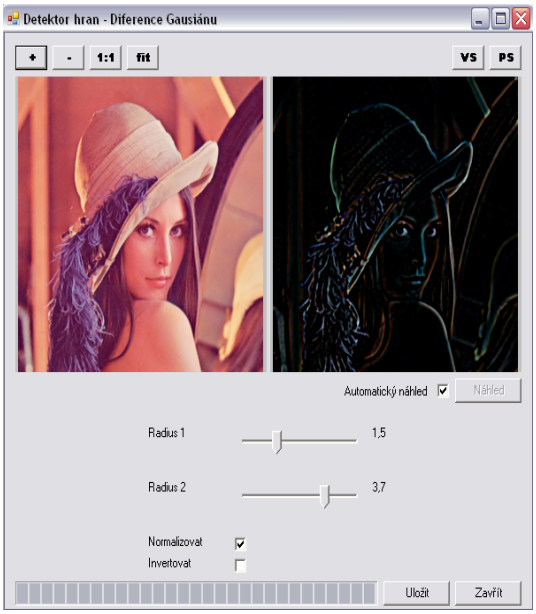
Obrázek 51 – Výsledné okno implementace operace Laplacian of Gaussian

pro Gaussovo vyhlazovaní se nastavuje pomocí posuvníku `Sigma`. V některých případech je dobré po aplikaci hranového detektoru, roztáhnout kontrast obrazu (normalizovat obraz) a tím získáme kontrastní hrany. To provedeme zaškrtnutím políčka `Normalizovat`. Další možností, která je k dispozici, je invertování výsledného obrazu. To je zajištěno zaškrtnutím políčka `Invertovat`. Výsledek implementace LoG je ukázán na obrázku 51.

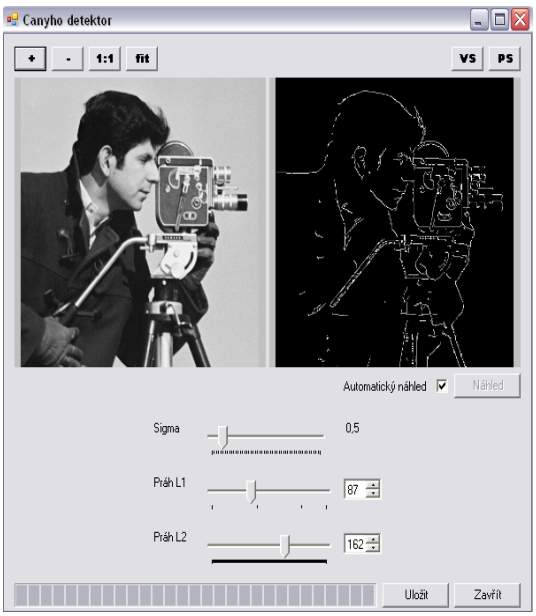
Další implementovanou metodou je Diference Gaussiánu (DoG). Již znázvu je patrné, že jde o rozdíl dvou vyhlazených obrazů Gaussiánem s různou hodnotou směrodatné odchylky $sigma$. Výsledek je zobrazen na obrázku 52. Podává také výborné výsledky. Zobrazuje pouze významné hrany, jelikož nevýznamné jsou odstraněny vyhlazením.

9.5.1 Cannyho hranový detektor

Poslední testovaný a nejlepší hranový detektor je Cannyho detektor. V okně Cannyho detektoru, které můžete vidět na obrázku 53, je několik posuvníků, kterými můžete volit různá nastavení pro detektor. Prvním je posuvník `Sigma`,

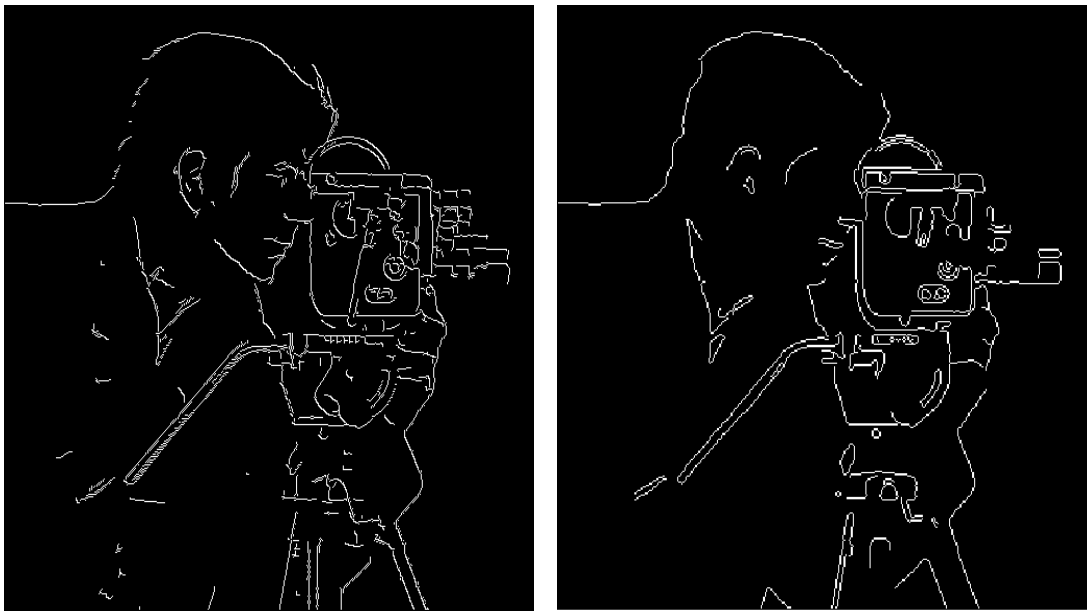


Obrázek 52 – Výsledné okno implementace operace Diference Gaussiánu



Obrázek 53 – Výsledné okno pro Cannyho detektor

kterým se nastavuje směrodatná odchylka Gaussova vyhlazování (viz kapitola 7.2.1.2). Druhým je práh $L1$ a třetím práh $L2$. Tyto dvě hodnoty nastavují nízký a vysoký práh pro prahování s hysterezí jak je uvedeno v kapitole 7.3.3



Obrázek 54 – Porovnání výsledků Cannyho detektoru hran a) Cannyho detektor hran, b) Cannyho detektor hran z *Matlabu*

Bohužel porovnání s výsledkem z programu *Matlab* (viz obr. 54), nedopadlo příliš dobře. Rozdíly jsou patrné na první pohled. Třetí část Cannyho algoritmu je implementována hůře než v *Matlabu*. Při implementaci jsem využil ten nejjednodušší způsob, který nemá dostatečné výsledky. Další rozdíl mezi algoritmy je v hledání hran. *Matlab* vyhledává hrany pomocí operace Laplacian of Gaussian, který podává lepší výsledky než mnou implementovaný Sobelův operátor. Funkce, pro testování hranových detektorů, napsaná v programu *Matlab* je ukázána v algoritmu 9.

Algoritmus 9 Funkce v *Matlabu* pro hledání hran různými detektory

```
function HranovyDetektor(cesta, metoda, prah)
   % parametr cesta -> urcuje cestu k souboru
   %    - podporovane obrazy jsou 8bitove v odstinech sedi
   % parametr metoda -> urcuje druh detektoru
   %    - mozne hodnoty:
   %         'roberts', 'sobel', 'prewitt',
   %         'zerocross', 'log', 'canny'
   % parametr prah -> urcuje hodnotu prahu
   %    - zadane jako vektor [low high]

   vstup = imread(cesta); % nacteni vstupniho obrazu
   figure, imshow(vstup); % zobrazeni vstupniho obrazu

   % test na existenci promenne prah
   if exist('prah') == 0
      vystup = edge(vstup, metoda);
   else
      vystup = edge(vstup, metoda, prah);
   end

   figure, imshow(vystup); % zobrazeni vystupniho obrazu
```

10 Závěr

Cílem práce byla implementace metod předzpracování obrazu, které mají sloužit jako podpora při výuce předmětů Počítačová grafika a Zpracování obrazu.

V teoretické části byla pečlivě popsána a rozebrána oblast předzpracování obrazu. Jedná se o část zpracování obrazu, která má za úkol vylepšit obraz před dalším zpracováním. Jedná se o metody jasové transformace, operace s histogramem, geometrické transformace a filtrace obrazu. Kvůli obsáhlosti tématu, jsou popsány pouze metody z diskrétní oblasti.

V praktické části jsou ukázány výsledky popsaných metod, které jsou porovnávány s odpovídajícími metodami z programu *Matlab*. Na závěr většiny metod je shrnutí, které popisuje vhodnost užití v praxi.

Celá kapitola předzpracování obrazu je příliš obsáhlá na to, aby ji jedinec dokázal popsat celou sám, proto vznikl požadevek na jednoduchou rozšiřitelnost programu. Splněním tohoto požadavku, pomocí přidávání modulů, je možné, aby program byl jednoduše rozšířen o další metody předzpracování obrazu jako jsou:

- Morfologické operace,
- zpracování obrazu ve frekvenční oblasti,
- segmentace obrazu a mnoho dalších.

Poté se rozšířený program bude moci stát plnohodnotnou pomůckou při výuce předmětu Zpracování obrazu a může být i vstupním bodem do dalších oblastí vývoje, jako je zpracování videa, apod.

11 Literatura

[1] ŽÁRA,V. *Moderní počítačová grafika.* 2. vyd. Praha, Cpress, 2005. 609 s. ISBN 80-251-0454-0.

[2] DOBEŠ, M. *Zpracování obrazu a algoritmy v C#.* Praha, BEN – Technická literatura, 2008. 144 s. ISBN 978-80-7300-233-6.

[3] HLAVÁČ, V., ŠOŇKA, M. *Počítačové vidění.* Praha, Grada, 1993. 272 s. ISBN 80-85424-67-3

[4] HLAVÁČ, V. SEDLÁČEK, M. *Zpracování signálů a obrazů.* 2. vyd. Praha, CVUT, 2006. 255 s. ISBN 978-80-01-03110-0

[5] GONZALEZ, Rafael. WOODS, Richard. *Digital image processing.* 3 rd Ed. Prentice Hall, 2002. 956 pages. ISBN 9780131687288.

[6] FRIBERT, Miroslav. *Základy zpracování obrazu.* Pardubice, Univerzita Pardubice, říjen 2006. 104 s. ISBN 80-7194-901-9

[7] MARTIŠEK, Dalibor. *Matematické principy grafických systémů.* Brno, Littera, 2002. 296 s, ISBN 80-85763-19-2

[8] BRUCHANOV, Martin. *Diskrétní 2D konvoluce* [online]. ČVUT Praha, 2006, URL: <http://bruxy.regnet.cz/fel/36ACS/konvoluce.pdf>

[9] LINKA, A., VOLF, P., KOŠEK, M. *Zpracování obrazu a jeho statistická analýza* [online]. Publikováno 6. 5. 2004, [cit. 2009-07-20]. URL: <http://www.e-learning.tul.cz/cgi-bin/elearning/elearning.fcgi?ID_tema=67&stranka=publ_tema>.

[10] TANG, K. *Image noise - Techniques for Image Processing and Classifications in Remote Sensing* [online]. Vystaveno 2004 [cit. 2009-07-30]. URL: <http://www.dig.cs.gc.cuny.edu/seminars/IPCV/pres12.pdf>

[11] FUNKHOUSER, T. *Image Warping* [online]. Princeton University (New Yersey), 2000, URL: <http://www.cs.princeton.edu/courses/archive/fall00/cs426/lectures/warp/warp.pdf>

[12] FISHER, R. PERKINS, S. WALKER, A. WOLFART, E. *Laplacian/Laplacian of Gaussian* [online]. Vystaveno 2003 [cit. 2009-07-24]. URL: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/log.htm>

[13] MORSE, B. *Edge Detection* Brigham Young University, 2000 ©1998-2000, last modified 2000-2-12. URL: <http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/MORSE/edges.pdf>

[14] FISHER, R. PERKINS, S. WALKER, A. WOLFART, E. *Unsharp Filter* [online]. 2003, last revision 2004-3-1 [cit. 2009-8-5] URL: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/unsharp.htm>

[15] MCHUGH, Sean. *Cambridge in color - Photography tutorial and Gallery.* Cambridge university, 2005. URL: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/unsharp.htm>

[16] BERÁNEK, Jan. *Metody detekce a reprezentace hran v obraze* [online]. Brno, 2007. Bakalářská práce na fakultě informačních technologií VUT v Brně, URL: <http://www.fit.vutbr.cz/study/DP/rpfile.php?id=3045>

[17] ALEXANDR, Lubomír. *Výuka počítačové grafiky cestou WWW* [online]. Brno, 1999. Diplomová práce na fakultě elektrotechniky a informatiky VUT v Brně, URL: <http://lubovo.misto.cz/_MAIL_/curves/obsah.html>

[18] NEVAŘIL, Lubomír. *Pravděpodobnostní převzorkování signálů* [online]. Praha, 2007. Odborná práce na fakultě jaderné a fyzikálního inženýrství ČVUT v Praze, URL: <http://dar.site.cas.cz/download.php?bd=546>

[19] DROBEK, Milan. *Charakteristiky a úpravy rastrového obrazu*, SOUT Přelouč, 2008, změněno 26. 12. 2008. URL: <http://www.sout-prelouc.cz/stranky/polygrafie_grafika_drobek/dokumenty/maturita/graf_6_rastr2charauprav.pdf>

[20] PIHAN, Roman. *Vše o světle – 13. Histogram* [online]. Vystaveno 18 . 4. 2007 [citováno 18. 6. 2009]. URL: <http://www.fotografovani.cz/art/fozak_df/rom_1_13_histogram.html>

[21] *Contrast streatching* [online]. 30. 3. 1999, URL: <http://www.geocities.com/ResearchTriangle/Thinktank/5996/techpaps/introip/manual03.html>

[22] HLAVÁČ, Václav. *Předzpracování v prostoru obrazu* [online]. Poslední úpravy 9. 4. 2009 [citováno 20.7.2009]. URL: <http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/21PredzpracObr.pdf>

[23] *Bilineární interpolace* [online]. 2007, poslední revize 28. 4. 2009 [citováno 12. 8. 2009]. URL: <http://cs.wikipedia.org/wiki/Bilineární_interpolace>

[24] WOLBERG, G. *Digital image warping*, IEEE Computer Society Press, California, 1990

[25] *Konvoluce - Wikipedie, otevřená encyklopedie* [online]. 2006, poslední revize 25. 05. 2009 [citováno 20. 7. 2009]. Dostupné z: <http:/cs.wikipedia.org/wiki/Konvoluce>.