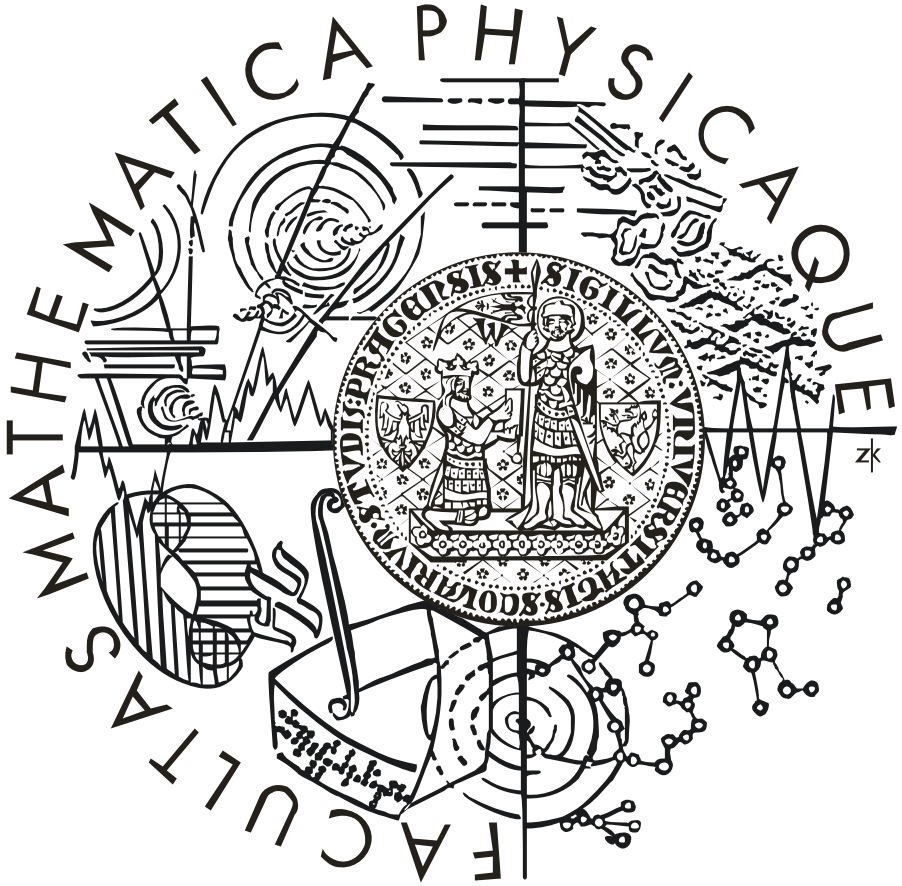
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Lukáš Hošek

Filtrace šumu z digitálních fotografií

Ústav formální a aplikované lingvistiky
Vedoucí bakalářské práce: Mgr. Pavel Machek
Studijní program: Informatika, programování

2007

Poděkování

Děkuji mému vedoucímu za jeho podporu při vývoji projektu a cenné rady při psaní programu i textu práce.

Prohlašuji, že jsem svou bakalářskou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 5.8.2007 Lukáš Hošek

Obsah

Název práce: Filtrace šumu z digitálních fotografií
Autor: Lukáš Hošek
Katedra (ústav): Ústav formální a aplikované lingvistiky
Vedoucí diplomové práce: Mgr. Pavel Machek
e-mail vedoucího: Pavel.Machek@mff.cuni.cz
Abstrakt:
Cílem práce je nastudovat a implementovat nejčastěji používané algoritmy na filtrovaní šumu z obrazů. Cílem by měly být nástroje na:

- filtrování aditivního šumu - nastudovat a implementovat alespoň dva netriviální algoritmy
- odstraňování vad obrazu způsobených defektními pixely na snímacím čipu ("hot pixels")

Výstupem bude modul do programu gimp a samostatně volatelná konzolová aplikace.
Klíčová slova: digitální fotografie, filtrování šumu, filtry zachovávající hrany

Title: Removal of noise from digital photos
Author: Lukáš Hošek
Department: Institute of Formal and Applied Linguistics
Supervisor: Mgr. Pavel Machek
Supervisor's e-mail address: Pavel.Machek@mff.cuni.cz
Abstract:
The goal of this work is to study and implement most frequently used algorithms for noise filtering. The results should be tools for:

- filtering additive noise – study and implement at least two nontrivial algorithms
- removal of image defects caused by defective pixels on image sensors ("hot pixels")
Keywords: digital photography, noise filtering, edge perserving filters

1. Úvod

1.1. Existující projekty a přínos tohoto projektu

Tématem této bakalářské práce je odstraňování vad digitálních fotografií způsobených defekty při výrobě snímacích čipů a nepřesnostmi inherentně spjatými se snímacím procesem. V podstatě každý program určený pro manipulaci s obrazem nebo správu fotoalb obsahuje nějakou více či méně pokročilou formu filtrace šumu.

K problému filtrování šumu se dá přistupovat mnoha způsoby. Nejjednodušší metoda je zřejmě *plovoucí průměr* (*box filter*). Při tomto postupu je hodnota každého pixelu na rekonstruovaném snímku nahrazena průměrem hodnot z okolí. *Gaussovský filtr* pracuje podobně, s tím rozdílem, že okolním pixelům je přiřazena různá váha odpovídající gaussovskému prostorovému rozložení. Gaussovský filtr účinně filtruje šum a produkuje vizuálně příjemnější výsledky než box filter (gaussovské rozložení frekvencí vede k dobrému ("optimálnímu") poměru mezi lokalizací v prostorové a frekvenční doméně), navíc se dá pomocí FFT efektivně implementovat, a proto je dnes součástí většiny metod na filtrování šumu. Gaussovský i box filtr jsou založené na principu lineární kombinace okolních pixelů – takovým filtrům se obecně říká *lineární filtry*. Objevily se snahy vylepšit dále koncept lineárního filtru pomocí optimálnějšího rozložení vah (například [1], kde se pokoušeli najít optimální filtr pomocí gradientní metody), ale ukazuje se, že jiná rozložení vah než gaussovské o mnoho lepší výsledky nedávají.

Problémem lineárních filtrů je, že nezachovávají hrany a detailní kresbu. Proto se také gaussovský filtr nejčastěji používá pro rozmazávání obrazu.

Druhá kategorie filtrů jsou *statistické filtry*. Do této kategorie spadají filtry pracující s mediánem lokálního okolí. Tyto filtry obecně zachovávají hrany lépe než lineární filtry, ale pořád mají problémy s rohy a dvourozměrnými rysy. Mediánové filtry dobře filtrují impulsní šum (šum, ve kterém poškozený pixel nenese žádnou informaci o původní hodnotě). Součástí této práce je implementace mediánového prahovacího filtru. Existuje spousta prací, které se pokoušely vylepšit mediánový filtr, například vážený mediánový filtr [2], kde se počítá medián ze seznamu, ve kterém jsou blížeji ležící pixely obsaženy vícekrát. Tento filtr zachovává hrany tím lépe, čím větší váha náleží blížeji ležícím pixelům, ale schopnost filtrovat šum se úměrně s tím zmenšuje. Ke statistickým metodám patří také například *K-nearest neighbor operator* (viz [3]). V této metodě jsou hodnoty okolních pixelů setříděny a z *k* nejbližších sousedů původní hodnoty se počítá průměr.

Jiná skupina filtrů zachovávajících hrany jsou filtry, které se pokoušejí nějakým způsobem detekovat hrany nebo ostré gradienty – například pomocí adaptace tvaru konvolučního jádra na lokální gradient.

Větší část této práce se zabývá problematikou filtrů zachovávajících hrany. Implementoval jsem dva algoritmy, z nichž jeden dává výsledky srovnatelné s profesionálními komerčními programy na filtrování šumu, jako je například Noise Ninja [4] nebo NeatImage [5].

Druhou částí této práce je návrh a implementace algoritmu na odstraňování vad obrazu, které se projevují přepálenými pixely při dlouhých expozicích. Tento problém se sice dá řešit pomocí obecných postupů a nástrojů na filtrování impulsního šumu, ale metoda použití kontrolního snímku, kterou navrhuji já, je efektivní a jednoduše implementovatelná.

1.2. Stručný průvodce po bakalářské práci

První kapitola obsahuje přehled předchozích prací o filtrování šumu a shrnutí hlavních cílů a výsledků mé práce.

Druhá kapitola je věnována stručné analýze charakteristik šumu v digitálních fotografiích.

Třetí kapitola je věnována analýze a implementaci algoritmů navržených J. van de Weijerem a R. van den Boomgaardem v práci [6]. Popisuje princip fungování algoritmů filtrování šumu pomocí lokálního modu a generalizovaného prostorově-tonálního gaussovského filtru, a uvádí kvantitativní i kvalitativní výsledky implementovaných algoritmů.

Čtvrtá kapitola se krátce věnuje mediánovému prahovacímu filtru.

Pátá kapitola popisuje můj navržený postup pro filtrování *hot pixels* z fotek focených s dlouhou expoziční dobou.

V šesté kapitole naleznete uživatelskou a programátorskou dokumentaci ke všem implementovaným programům a modulům.

Sedmá kapitola obsahuje popis přiloženého CD.

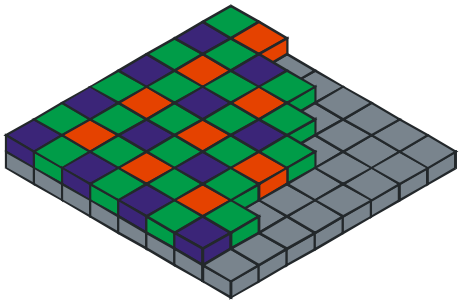
Osmá kapitola obsahuje závěrečné shrnutí.

2. Analýza šumu v digitálních fotografiích

2.1. Princip fungování CCD snímačů

CCD čip sám o sobě je analogový posuvný registr – zařízení, které uchovává navzorkovaný analogový signál a umožňuje ho postupně přenášet přes soustavu kondenzátorů. Původní analogový signál se získává z fotosenzorů, které jsou v CCD snímačích uspořádané do obdélníkové mřížky a překryté barevnými filtry. Zdaleka nejčastěji používaným rozložením barev na filtru je Bayerova maska (viz obr. 1). V Bayerově masce jsou pixely sdružené do skupin po čtyřech, v každé skupině je jeden modrý, jeden červený a dva zelené pixely (lidské oko je na zelenou nejcitlivější). Z toho vyplývá, že v každém pixelu je sice informace o jasu, ale informace o chromaticitě pixelu je hrubší a musí se interpolovat (tomuto procesu se říká *demosaicing*). Na subjektivní vjem obrazu to nemá velký vliv, protože lidské oko je na informaci o chromaticitě méně citlivé, nicméně je to důležité z hlediska analýzy charakteru šumu, protože proces interpolace vnáší do zašumění obrazu jistou korelaci mezi okolními pixely a hodnoty zašumění v sousedních pixelech tedy nejsou nezávislé náhodné veličiny.

Náboj, který je po expozici snímku v kondenzátorech u každého pixelu, je výsledkem fotoionizace způsobené fotony interagujícími se senzory. Náboje jsou poté v CCD čipu periodicky posouvány a čteny na výstupních elektrodách. Výstupní signál na elektrodách je příliš slabý na to, aby mohl být dál přímo zpracováván, a musí být nejprve zesílen. Zesilovač by měl mít dostatečný zisk na to, aby další zdroje analogového šumu byly pro výsledek nevýznamné.



Obrázek 1: Bayerova maska

2.2. Zdroje šumu

Podle [7] se dá šum v digitálních fotografiích modelovat takto:

Materiály, ze kterých jsou vyrobena dielektrika v CCD snímačích nejsou pochopitelně dokonalé izolanty, takže už v senzorech dochází k úniku proudu. Ze stejné příčiny je do signálu zanesen další šum v zesilovači a konečně kvantizační šum je rozdíl mezi původním analogovým a kvantifikovaným digitálním signálem. Tyto tři zdroje šumu se dají dohromady modelovat jako bílý gaussovský šum.

Dalším zdrojem šumu je samotný mechanismus tvorby náboje. Pravděpodobnost, že elektron vytvoří pár s elektronovou dírou je nezávislá na interakci s předchozími elektrony, dá se tedy předpokládat, že výsledná ionizace má poissonovské rozdělení. Pokud je výsledkem expozice n párů elektron-díra v jednom pixelu za jednu jednotku času, dá se očekávat rozptyl n. To znamená, že bez ohledu na kvalitu snímacího zařízení je jistá míra šumu ve fotografii nevyhnutelná už z fyzikální podstaty snímacího procesu. Pokud bychom fotili jednobarevnou scénu a měli k dispozici ideální CCD senzor vyrobený z dokonalých materiálů, přesto bychom po expozici zaznamenali na výsledném snímku šum, jehož intenzita by rostla s jasem scény. Nicméně z měření [7] vyplývá, že tento zdroj šumu je ve srovnání s ostatními zanedbatelný a dominantní složku tvoří elektronický šum. Navíc je důležité uvědomit si, že tento druh šumu vnímá i lidské oko, protože excitace světločivných buněk na sítnici je poissonovský proces stejně jako tvorba náboje na fotoionizačních senzorech. Poissonovská podstata zrakového vjemu je to, co nám znemožňuje vidět za dne hvězdy na obloze. Jistá míra zašumění je tedy pro lidi přirozená a tento zdroj šumu má

narozdíl od elektronického šumu pro lidské oko přirozenou charakteristiku (a je nesrovnatelně slabší). Ve zbytku práce si tedy dovolím tento zdroj šumu zanedbat.

Současné CCD snímače trpí ještě jedním neduhem: fotosenzitivní křemíkový materiál je také citlivý na náboj. To se při normálních expozičních dobách neprojevuje, ale kdybychom mohli exponovat snímek dostatečně dlouho, i v naprosté tmě by kvůli tomu nakonec byly všechny pixely vysvícené. Bohužel nedokonalosti ve výrobním procesu způsobují, že rychlost úniku náboje se pixel od pixelu liší a u některých pixelů na snímači je výrazně vyšší. Vlivem toho se na snímcích exponovaných s dlouhou expoziční dobou (od jedné sekundy výš, podle kvality snímače) projevuje jev označovaný jako *hot pixels* – několik subpixelů na snímku je výrazně více vysvíceno. Charakteristika tohoto jevu se nejvíce podobá impulsnímu šumu a filtry navržené pro filtrování impulsního šumu také dokáží tento jev účinně filtrovat. Nicméně součástí této práce je návrh a implementace jiného postupu, který více využívá specifické vlastnosti tohoto jevu.

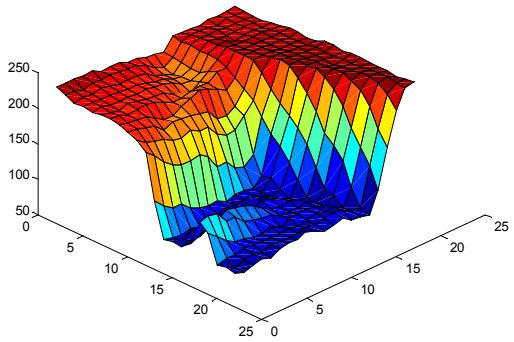
3. Filtry zachovávající hrany

V první kapitole jsem se zmínil o tom, že gaussovský filtr je pravděpodobně nejpoužívanější filtr na odstraňování šumu. Podívejme se nejdřív, jak funguje a v čem spočívají jeho nevýhody.

3.1. Klasický gaussovský filtr

Gaussovským filtrováním obrazu f definovaného obrazovou funkcí $f : R^n \to R$ rozumíme konvoluci f gaussiánem. Výsledkem je obraz g definovaný takto (s explicitní normalizací jádra):

$$g(x_0) = \frac{\int_{R^n} f(x)G(x - x_0, \sigma)dx}{\int_{R^n} G(x - x_0, \sigma)dx}$$

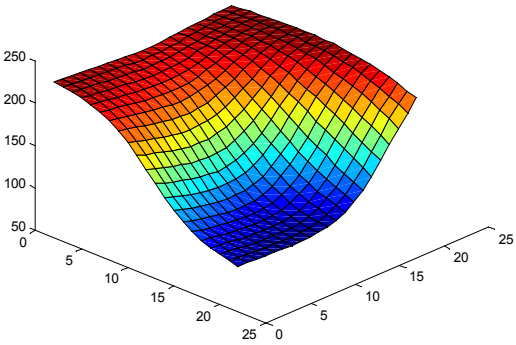


Obrázek 2: Okolí bodu s ostrým přechodem

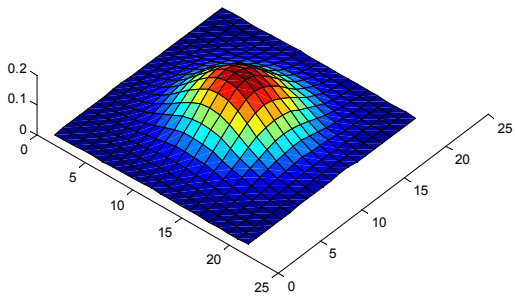
Kde gaussián G je funkce ve tvaru

$$G(x, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \cdot e^{\frac{-x^T \times x}{\sigma^2}}$$

Tvar gaussiánu je vidět na obrázku 4. Fourierova analýza gaussovského filtru ukazuje, že jeho funkce spočívá v utlumení vyšších frekvencí obrazové informace. To je naprosto legitimní způsob odstraňování šumu, který fyzici používají odedávna. Narozdíl od fyzikálních aplikací se tento postup ale pro filtrování šumu z digitálních fotografií nehodí z jednoho prostého důvodu: Ve fyzikálních měřeních se často nachází zajímavá informace v úplně jiném pásmu než šum, a utlumení zašuměných frekvencí tedy informaci nijak neznehodnotí. To ale pro fotky neplatí, protože ve vyšších frekvencích, ve kterých se obrazový šum nachází, je zároveň obsažena detailní kresba fotografie. Na obrázku 3 je dobře vidět výsledek utlumení vyšších frekvencí v obrazové informaci: šum je sice pryč, ale z hran se staly pozvolné přechody. Tento problém přitom nelze řešit volbou jiného konvolučního jádra, protože statický charakter filtrovacího mechanismu lineárních způsobuje, že do výsledné barvy pixelu je započítána barva z okolních pixelů, které přitom patří do jiného regionu. V následujících dvou kapitolách rozebereme postupy navržené J. van de Weijerem a R. van den Boomgaardem v [6].



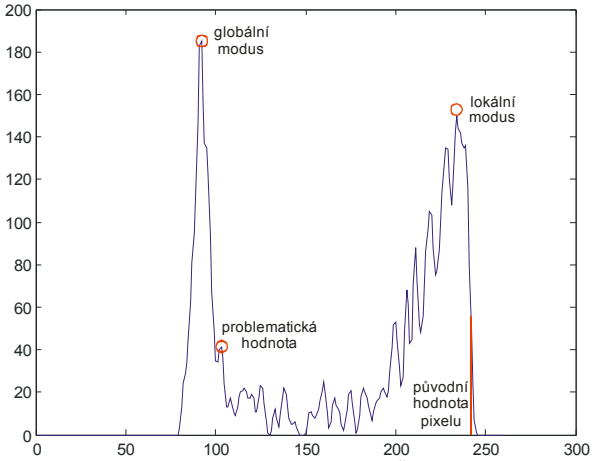
Obrázek 3: Ostrý přechod
po aplikaci gaussovského filtru



Obrázek 4: Gaussovské konvoluční jádro

3.2. Lokální modus

Na obrázku 5 vidíme histogram okolí bodu s ostrým přechodem z obrázku 2. můžeme jasně vidět dvě špičky, které odpovídají hornímu a dolnímu regionu obrázku. Jedna z nich je zároveň globální modus, ale původní hodnota pixelu leží v části histogramu, která patří spíše k druhé špičce, označené *lokální modus*.



Obrázek 5: Histogram ostrého přechodu

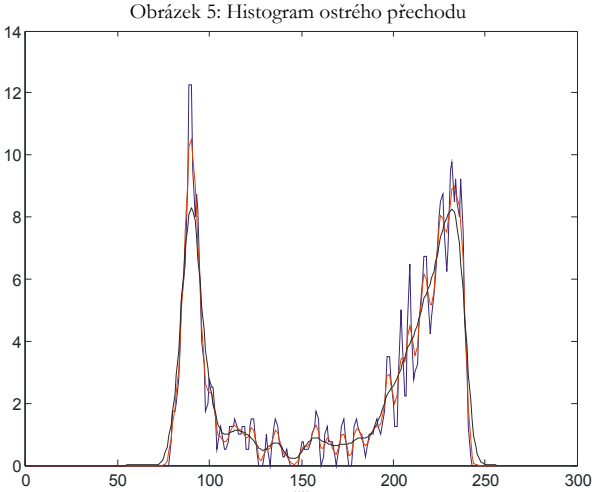
Hledání lokálního modu je podstata této metody. Nejdříve definujeme pojem *vážený lokální histogram*. Hodnota váženého lokálního histogramu v bodě *i* s počátkem v bodě x_0 a senzorovým měřítkem 0 je definován jako

$$H(x_0, i, 0) = \delta(i - f(x))$$

Kde δ je Diracův puls. Vážený histogram s nenulovým senzorovým měřítkem je potom definován takto:

$$H(x_0, i, \sigma) = H(x, i, 0) \otimes_x G(x, \sigma)$$

Neboli pixely, které jsou dál od středu, dostávají v histogramu menší váhy, přičemž váha v histogramu odpovídá gaussiánu prostorové vzdálenosti.


Obrázek 6: vliv tonálního měřítka na hladkost histogramu

V reálné implementaci bychom ovšem narazili na problém, pokud bychom v takto definovaném histogramu chtěli hledat lokální modus. Pokud se podíváme na obrázek 5 a zkusíme najít lokální modus pixelu s původní hodnotou označenou na grafu jako *problematická hodnota*, zjistíme, že už se v lokálním modu nacházíme, přestože tušíme, že pro filtrování by bylo výhodnější překonat lokální pokles a dostat se až do špičky označené *globální modus*. Proto použijeme kromě prostorového ještě tonální filtrování: hodnotu v bodě *i* váženého tonálně filtrovaného lokálního histogramu s počátkem v x_0, se senzorovým měřítkem σ_s a s tonálním měřítkem σ_t definujeme takto:

$$H(x_0, i, \sigma_s, \sigma_t) = H(x_0, i, \sigma_s) \otimes_i G(i, \sigma_t)$$

Obrázek 6 ukazuje vliv stoupajícího σ_t na hladkost histogramu.


Obrázek 7:(1) původní snímek, (2) snímek degenerovaný 25% nekorelovaným gaussovským šumem (RMSE 38,16), (3) snímek po rekonstrukci s $\sigma_s = 5$, $\sigma_t = 5$ (RMSE 23,43), (4) snímek po rekonstrukci s $\sigma_s = 10$, $\sigma_t = 10$ (RMSE 24,12)

První výsledky ukazují, že filtr dokáže úspěšně redukovat hladinu šumu. V obou dvou rekonstruovaných snímcích je střední kvadratická chyba menší než v zašuměném snímku, přičemž detaily zůstaly zachovány. Agresivnější nastavení filtru (4) vede k výsledku, který je sice méně zašuměný, ale přesto, že ostré přechody jsou zachovány, subjektivně postrádá texturu a vypadá umělohmotně.

Obrázek 8: (1) snímek degenerovaný 50% nekorelovaným gaussovským šumem (RMSE 74,38), (2) rekonstrukce s $\sigma_s = 5$, $\sigma_t = 10$ (RMSE 54,31), (3) rekonstrukce s $\sigma_s = 5$, $\sigma_t = 30$ (RMSE 27,46), (4) rekonstrukce s $\sigma_s = 5$, $\sigma_t = 40$ (RMSE 26,96), (5) rekonstrukce s $\sigma_s = 10$, $\sigma_t = 10$ (RMSE 37,71), (6) rekonstrukce s $\sigma_s = 10$, $\sigma_t = 30$ (RMSE 32,40)

Výsledky testů na silněji degenerovaných snímcích ukazují více než cokoliv jiného, že střední kvadratická chyba nemá moc velkou vypovídací hodnotu o kvalitě rekonstrukce – přestože papírově bylo nejlepší nastavení $\sigma_s = 5$, $\sigma_t = 40$, subjektivně působí snímek rekonstruovaný s parametry $\sigma_s = 5$, $\sigma_t = 30$ přirozeněji, protože obsahuje detailnější kresbu vlasů.

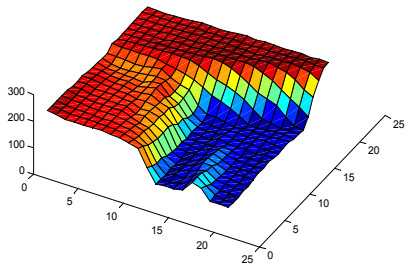
Přestože z výsledků vyplývá, že zde popsaná a implementovaná metoda dokáže účinně filtrovat šum a přitom zachovávat detaily, k ideálnímu výsledku má ještě daleko. V první řadě je problematická rychlost implementace – gaussovský filtr lze implementovat pomocí násobení ve frekvenční doméně s časovou složitostí $O(n^2 \log n)$, ale složitost filtrování pomocí lokálního modu je $O(n^4)$. Druhým problémem je korelace mezi barevnými kanály. Teoreticky lze snadno popsat, jak by vypadalo filtrování pomocí lokálního modu, které by zpracovávalo všechny kanály najednou a hledalo by lokální modus vícerozměrného histogramu, prakticky je ale obtížné mít histogram se třemi barevnými složkami a 256 hodnotami pro každou složku – není sice problém udržovat v paměti 16 milionů hodnot, ale konvolvovat takový histogram pro každý pixel trojrozměrným gaussiánem by bylo časově neúnosné. Třetí problém je samotný charakter filtru – lokální medián je příliš "tvrdá" funkce a pro agresivnější nastavení produkuje nepřirozené obrazy.

Většinu těchto problémů ale řeší další popsaná metoda, která dokáže využívat korelace mezi kanály a dává velmi dobré a přirozeně vypadající výsledky. Navíc lze

teoreticky dokázat, že iterováním této metody dostaneme výsledek konvergující k filtrování lokálním modem (viz [6]).

3.3. Generalizovaný prostorově – tonální gaussovský filtr

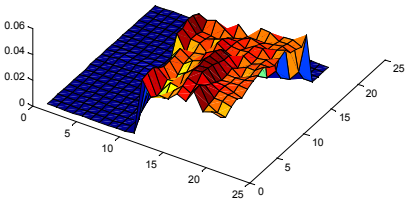
Jak jsem uvedl výše, gaussovský filtr má některé zajímavé vlastnosti – často bývá popisován jako optimální kompromis mezi lokalizací v prostorové a frekvenční doméně a subjektivně netrpí artefakty typickými pro box filter nebo cylindrický filtr. Jediný problém, který brání jeho úspěšnému použití pro filtrování šumu z obrázků je, že tlumí *všechny* vysokofrekvenční informace. Následující metoda se tento problém snaží řešit pomocí adaptace tvaru konvolučního jádra na tonální informaci v lokálním okolí bodu.



Obrázek 9: Okolí bodu

Prostorově – tonální filtr je přímočará generalizace klasického gaussovského filtru navržená v [6]. Původní vzorec

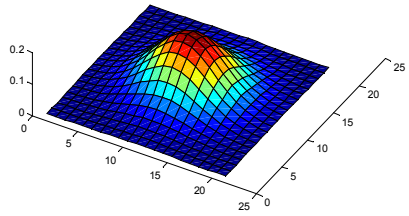
$$g(x_0) = \frac{\int_{R^n} f(x)G(x - x_0, \sigma)dx}{\int_{R^n} G(x - x_0, \sigma)dx}$$



Obrázek 10: Tonální jádro

je rozšířen tak, že váha okolních pixelů na obrázku je určena nejen jejich prostorovou vzdáleností, ale také rozdílem hodnot (neboli tonální vzdáleností):

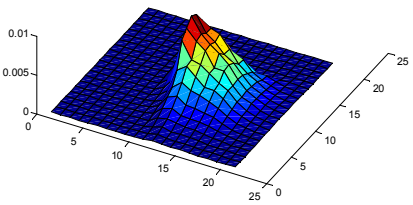
$$g(x_0) = \frac{\int_{R^n} f(x)G(x - x_0, \sigma_s)G(|f(x) - f(x_0)|, \sigma_t)dx}{\int_{R^n} G(x - x_0, \sigma_s)G(|f(x) - f(x_0)|, \sigma_t)dx}$$



Obrázek 11: Gaussián

Povšimněte si, že tento předpis bez jakýchkoliv úprav funguje i pokud je obrazová funkce *f* zobrazením do vícerozměrného barevného prostoru.

Podíváme se na názornou ukázku funkce tohoto filtru: na obrázku 9 je znázorněno okolí bodu, který sám leží v regionu s nižšími (modrými) hodnotami. Obrázek 10 znázorňuje tonální jádro takového bodu vzhledem k jeho okolí. Na obrázku jde názorně vidět, že tonální jádro výrazně upřednostňuje hodnoty ležící ve stejném regionu jako původní pixel. Hodnota σ_t udává strmost poklesu přidělené váhy vzhledem k tonální vzdálenosti. Obrázek 11 znázorňuje klasický gaussián – ten je s tonálním jádrem složku po složce vynásoben a výsledkem je jádro konvoluce, které je znázorněno na obrázku 12. Na obrázku 12 jde názorně vidět, jak tento filtr pracuje – upřednostňuje bližší hodnoty před vzdálenějšími podobně jako gaussovský filtr, ale zároveň téměř nepoužívá hodnoty z jiných regionů.



Obrázek 12: Výsledné konvoluční jádro

Časová složitost tohoto algoritmu je $O(n^4)$ - pro každý pixel z původního snímku se musí z okolí spočítat nové jádro a to následně znormalizovat a konvolvovat s okolím, nicméně pečlivou optimalizací se dá dosáhnout na současných strojích časů kolem 20 sekund na fotografii.

12% gaussovský šum	
σ_s/σ_t	RMSE
bez filtrace	9,22
1/10	6,56
2/5	7,73
2/10	5,63
2/12	5,44
2/20	6,38
3/10	5,64

25% gaussovský šum	
σ_s/σ_t	RMSE
bez filtrace	37,97
2/20	17,21
3/20	16,50
3/30	13,81
3/35	13,92
3/40	14,42
4/30	14,74

50% gaussovský šum	
σ_s/σ_t	RMSE
bez filtrace	74,05
2/70	25,56
3/60	25,86
3/70	24,50
3/80	24,16
3/85	24,18
4/40	35,44
4/50	29,38
4/60	26,67
4/70	25,86
4/80	25,97

100% gaussovský šum	
σ_s/σ_t	RMSE
bez filtrace	99,14
4/100	51,20
4/150	40,48
4/200	38,13
4/250	37,70
5/100	51,33
6/150	42,91

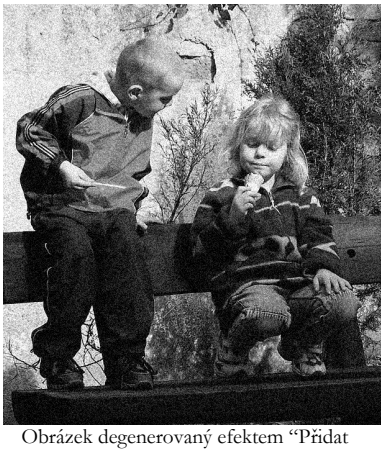
Tabulka ukazuje výsledky filtrování šumu generalizovaným prostorově – tonálním filtrem. Původní obrázek byl známý snímek Lenny. Opět, obzvláště u 100% zašumění, se ale ukazuje, že lidské oko nevnímá kvalitu snímku prostřednictvím střední kvadratické chyby – tak se například může zdát, že zvyšování tonální tolerance při vyšších zašuměních bude produkovat lepší výsledky, ve skutečnosti ale budou výsledky, ve kterých je sice silnější zbytkový šum, ale více detailů, vypadat subjektivně lépe.

Originální snímek | snímek degenerovaný 50% gaussovským šumem | rekonstrukce $\sigma_{s=5}, \sigma_{t=50}$

rekonstrukce $\sigma_{s=5}, \sigma_{t=60}$ | rekonstrukce $\sigma_{s=5}, \sigma_{t=80}$ | rekonstrukce $\sigma_{s=6}, \sigma_{t=50}$

Filtr Remove noise z Corel PhotoPaint | Filtr Despeckle z GIMPu | Filtr Selective Gaussian blur z GIMPu

Obrázek 13: Výsledky filtrování černobílého snímku Lenny s různými parametry a porovnání s výsledky jiných nástrojů

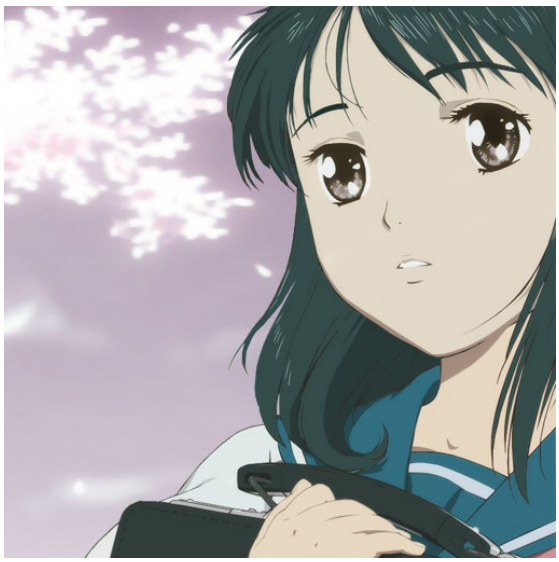


Obrázek degenerovaný efektem "Přidat zrno" z Zoner PhotoStudio

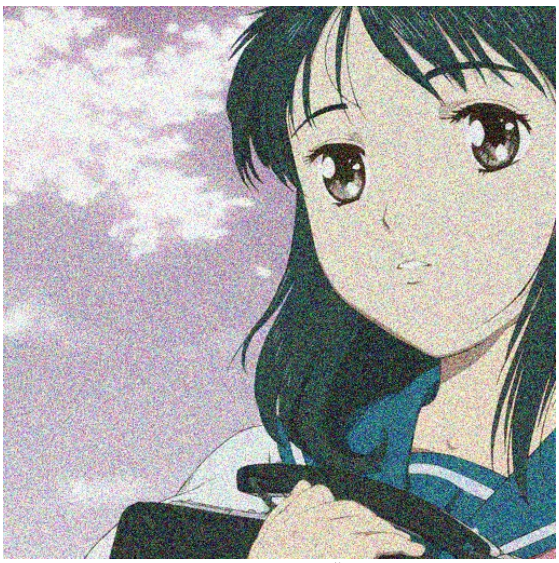
rekonstrukce $\sigma_{s=5}, \sigma_{t=50}$

rekonstrukce $\sigma_{s=6}, \sigma_{t=60}$

Obrázek 14: výsledky na obrázcích s negaussovskou charakteristikou zašumění

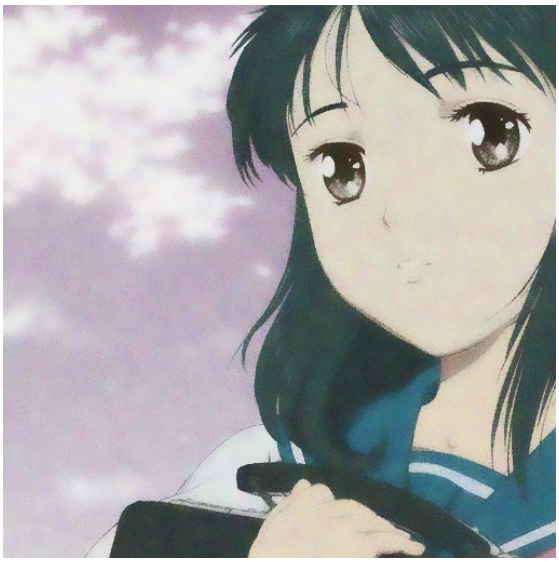


Originál



100% nekorelovaný gaussovský šum (RMSE 99,55)

rekonstrukce $\sigma_{s=5}, \sigma_{t=80}$ (RMSE 50,22)



rekonstrukce $\sigma_{s=7}, \sigma_{t=80}$ (RMSE 50,59)

Obrázek 15: Silně zašuměný obrázek s minimem detailní kresby

Výřez z původního snímku | degenerovaný snímek (RMSE 73,42) | rekonstrukce $\sigma_{s=5}, \sigma_{t=35}$ (RMSE 44,62)

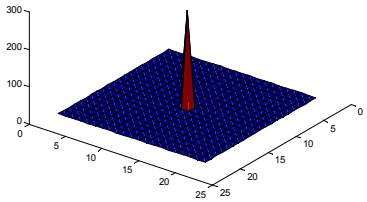
rekonstrukce $\sigma_{s=5}, \sigma_{t=40}$ (RMSE 46,65) | rekonstrukce $\sigma_{s=6}, \sigma_{t=35}$ (RMSE 45,50) | rekonstrukce $\sigma_{s=7}, \sigma_{t=45}$ (RMSE 51,42)

Obrázek 16: Výsledky rekonstrukce snímku obsahujícího velké množství drobných detailů degenerovaného 50% nekorelovaným gaussovským šumem

Výsledky ukazují, že filtr má některé pozoruhodné vlastnosti – detaily a hrany zachovává lépe než klasický gaussovský filtr, přičemž nastavení prostorového rozmazávání má na tuto vlastnost malý vliv. Filtr zachovává detailní kresbu a produkuje subjektivně přirozeněji vypadající výsledky než filtrování pomocí lokálního modu. Algoritmus dává výsledky lepší než filtry z grafických nástrojů typu Photoshop nebo PhotoPaint a srovnatelné s profesionálními nástroji na filtrování šumu typu NeatImage nebo Noise Ninja.

4. Mediánový prahovací filtr

Lineární filtry (jako klasický gaussovský filtr) dokáží účinně potlačit gaussovský šum, ale trpí jedním neduhem: malou odolností vůči impulsnímu šumu. To je demonstrováno na obrázcích 17 a 18 – jeden prudký výkyv v signálu dokáže po přefiltrování poškodit obrazovou informaci v celém okolí. To se dá vyjádřit pomocí konceptu *bodu zhroucení* [8]:



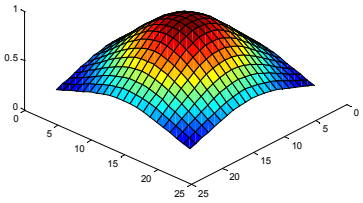
Obrázek 17: Špička signálu způsobená impulsním šumem

Mějme náhodné veličiny F, G. Prohorovova vzdálenost mezi těmito veličinami je definována jako

$$d(F,G) = \inf\left\{v \le 1 \middle| \forall A : F(A) \le G(A) + v\right\}$$

bod zhroucení v^* filtru $\left\{S_n \middle| n \ge 1\right\}$ je potom definován

$$v^* = \sup_{n\to\infty}\left\{v \le 1 \middle| \exists r_v : d(F,G) < v \Rightarrow G(\left\{\left|S_n\right| \le r_v\right\}) \to 1\right\}$$



Obrázek 18: Špička signálu po přefiltrování gaussovským filtrem

Lineární filtry mají $v^* = 0$, proto jeden výkyv může zničit informaci v celém okolí. Naproti tomu mediánový filtr má $v^* = 0{,}5$, takže pokud hustota impulsního šumu v okolí nepřekročí 50%, pracuje stále spolehlivě. Práce [4] ukazuje, že předfiltrování impulsního šumu před zpracováním lineárním filtrem umožní lépe zachovat obrazovou informaci na výsledném snímku. Autoři této práce navrhuje kombinaci mediánového filtru a lineárního filtru, jehož váhy byly natrénovány gradientní metodou. My máme pro druhou fázi navržen lepší filtr (popsaný v kapitole 3.3), ten je však stále založen na klasickém lineárním gaussovském filtru, a proto se dá při předfiltrování impulsního šumu očekávat zlepšení výsledků.

Impulsní šum je filtrován následujícím způsobem: okolím každého bodu je čtvercové okénko velikosti $(2n+1)\times(2n+1)$. Hodnoty z něj se setřídí a nalezne se medián. Pokud je rozdíl původní hodnoty pixelu a mediánu větší než práh *t*, je pixel považován za poškozený a jeho hodnota je na výstupu nahrazena mediánem. Jinak se použije původní hodnota pixelu.

(1) Originální snímek

(2) Snímek degradovaný impulsním šumem s 25% hustotou (RMSE 29,40)

(3) rekonstrukce, poloměr 2, práh 60 (RMSE 18,18)

(4) rekonstrukce, poloměr 5, práh 50 (RMSE 17,33)

(5) filtr GenGauss, $\sigma_{s=5}, \sigma_{t=40}$ (RMSE 35,73)

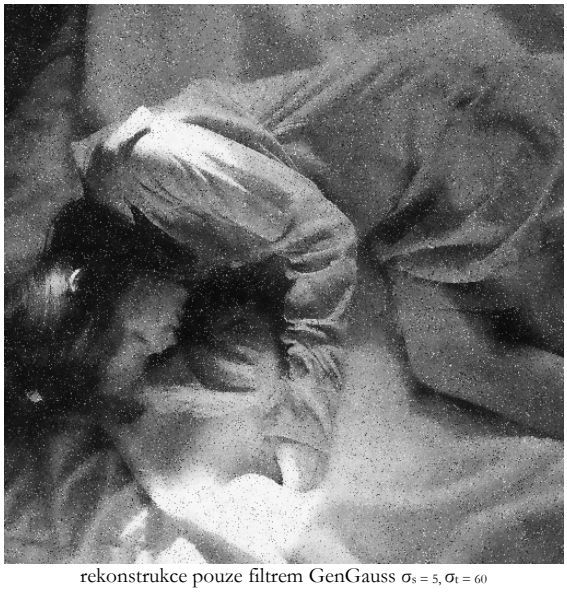
(6) filtr GenGauss, $\sigma_{s=5}, \sigma_{t=80}$ (RMSE 35,45)

Obrázek 19: Výsledky filtrování impulsního šumu

Na obrázku 19 jsou praktické výsledky filtrování. (5) a (6) jsou výsledky filtrování generalizovaným gaussovským filtrem z předchozí kapitoly. GenGauss se pro filtrování tohoto typu šumu nehodí – okolí poškozených bodů vnímá jako jiný region a pro jejich odfiltrování je potřeba nastavit σ_t příliš velké, takže je poškozena i původní informace o detailech. V obou případech použití filtru GenGauss vzrostla střední kvadratická chyba oproti nerekonstruovanému snímku. Za těchto podmínek dává mediánový prahovací filtr lepší výsledky; menší okolí pro hledání mediánu produkuje výsledky lépe zachovávající detaily. Takto zpracovaný snímek jde pak lépe dále zpracovávat jinými filtry. Na snímku (4) je výsledek příliš agresivního nastavení filtru – dívce na testovacím obrázku zmizelo oční víčko a ucho.

Šum v reálných případech je směsí gaussovského a impulsního šumu. Předfiltrování impulsního šumu umožní lépe ochránit původní obrazovou informaci.

snímek degenerovaný směsí gaussovského a impulsního šumu



rekonstrukce pouze filtrem GenGauss $\sigma_{s = 5}, \sigma_{t = 60}$

rekonstrukce pouze prahovacím mediánovým filtrem

rekonstrukce kombinací mediánového prahovacího filtru a filtru GenGauss

Obrázek 20: výsledky kombinovaného filtrování

5. Filtrování hot pixels

Ve druhé kapitole jsem popsal mechanismus vzniku jevu zvaného *hot pixels* – nepřesnosti ve výrobním postupu snímačů, které se projevují při delších expozičních dobách vysvícenými pixely. Hot pixels jsou ve své podstatě impulsní šum, dá se tedy filtrovat algoritmem popsaným v kapitole 4. Tento algoritmus sice filtruje impulsní šum, ale má některé nepříjemné vlastnosti (viz snímek 4 z obrázku 19, kde filtr odfiltroval dívce spolu s šumem i ucho).

Mediánový filtr klasifikuje pixely jako poškozené podle rozdílu oproti lokálnímu mediánu. Výsledkem této kapitoly je filtr, který klasifikuje pixely jako poškozené jinak, s méně falešnými pozitivními výsledky a odstraňuje tak méně detailů.

Pokud si prohlédneme sérii fotografií focených s dlouhou expoziční dobou během nějakého kratšího časového úseků, zjistíme, že přepálené pixely se objevují na stejných místech. Toho využívá mnou navržený postup: kromě zdrojového obrázku, který chceme filtrovat, pracuje navíc s *referenčním snímkem* – to je snímek tmy, který pořídíme se zakrytým objektivem a se stejným nastavením expozice jako původní snímek.

Původní idea byla, že referenční snímek stačí od zdrojového odečíst pixel po pixelu – výsledkem je ale obraz, kde je jeden typ artefaktu nahrazen jiným. Přepálené pixely bývají totiž na zdrojovém i referenčním snímku často vysvícené na maximum a jejich odečtením vznikne nula. Na výsledném snímku jsou pak místo přepálených pixelů černé díry.

Finální implementace tedy používá jiný postup: pixely referenčního snímku jsou porovnávány s prahovou hodnotou. Pokud hodnota na referenčním snímku práh překročí, znamená to, že na stejné pozici na zdrojovém snímku je poškozený pixel, který se pak rekonstruuje z okolních pixelů (lineárním filtrem, lokálním mediánem a podobně).

Výřez z originálního snímku | Filtrování pomocí referenčního snímku | Filtrování mediánovým prahovacím filtrem

Obrázek 21: Výsledky filtrování přepálených pixelů

Praktická implementace tohoto postupu musí řešit ještě jeden problém: degeneraci obrázku JPEG kompresí. Pokud fotoaparát ukládá fotografie ve formátu JFIF, i nízký stupeň komprese vede k zavlečení obrazových artefaktů – kolem přepálených pixelů se tvoří tmavší kroužky, které po filtraci ve výsledném snímku zůstávají. Moje implementace to řeší přefiltrováním referenčního snímku box filtrem s poloměrem 1 – hlavní algoritmus filtru tak označí i přilehlé pixely za poškozené.

Z výsledných obrázků jde vidět, že filtrace pomocí referenčního snímku produkuje lepší výsledky než mediánový prahovací filtr. Navrhovaný algoritmus odstranil na rozdíl od mediánového filtru všechny přepálené pixely a přitom neznehodnotil žádnou obrazovou informaci.

6. Dokumentace

Součástí práce byla praktické implementace všech výše popsaných algoritmů. Filtrování lokálním modem, generalizovaný prostorově-tonální filtr a mediánový prahovací filtr (v této kapitole také někdy označován jako impulsní filtr) byly nejprve implementovány jako skripty v MATLABu a následně jako pluginy pro grafický nástroj GIMP [9] a samostatné konzolové aplikace. Nástroj pro filtrování hot pixels pomocí metody popsané v kapitole 5 byl implementován jako samostatná grafická aplikace. Cílovou platformou všech programů jsou UNIXové operační systémy.

6.1. Uživatelská dokumentace

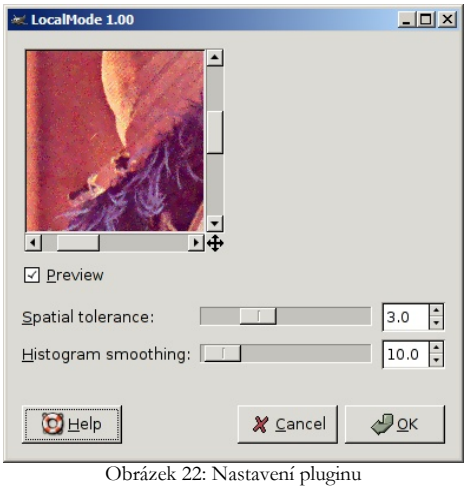
6.1.1. Filtrování pomocí lokálního modu – plugin do GIMPu

Systémové požadavky: pro instalaci je potřeba funkční toolchain a developerský balíček programu GIMP [9], který obsahuje knihovny potřebné pro překlad a nástroj *gimptool-2.0*.

Instalace: plugin je distribuován jako jediný zdrojový soubor *localmode*.c a instaluje se klasicky pomocí nástroje *gimptool*.

```
gimptool-2.0 --install localmode.c
```

Ovládání programu: po instalaci přibude v menu oken GIMPu s otevřeným obrázkem položka *Filters* ► *Enhance* ► *Local mode filter*... Po vyvolání této položky se objeví dialog s náhledem a možnostmi nastavení filtru vyobrazený na obrázku 22. Vlastnosti filtru je možné upravit pomocí parametrů *Spatial tolerance* a *Histogram smoothing*. *Spatial tolerance* nastavuje velikost okolí, ze kterého se počítá lokální vážený histogram. *Histogram smoothing* nastavuje velikost parametr σ gaussiánu, který se následné použije při filtrování histogramu. Podrobnější význam těchto parametrů je rozebrán v kapitole 3.2. V horní části okna je náhled, kde si lze prohlédnout výsledek před definitivním potvrzením volby. Tlačítkem *OK* se potvrdí nastavení filtru a spustí se výpočet. Tlačítko *Cancel* zavírá okno pluginu bez použití filtru.



Obrázek 22: Nastavení pluginu Local mode filter

6.1.2. Filtrování pomocí lokálního modu – konzolová verze

Systémové požadavky: pro instalaci je potřeba funkční toolchain a developerská verze knihovny *Imlib2* [10].

Instalace: konzolová verze filtru je distribuována jako zdrojový balíček *localmode-standalone.tar.bz2*. Rozbalení a překlad probíhá následujícím způsobem:

```
tar –xf localmode-standalone.tar.bz2
cd localmode-standalone/Release
make
```

Po skončení překladu je v adresáři *Release* spustitelný soubor *localmode*, který případně, pokud máte administrátorská práva, můžete pomocí programu *install* zkopírovat na vhodné místo v systému (*/usr/local/bin* apod.).

Ovládání programu: program se volá následujícím způsobem:

```
localmode -s spatial -t tonal infile outfile
```

Všechny parametry jsou povinné. *spatial* je desetinné číslo, které nastavuje velikost okolí, ze kterého se počítá lokální histogram. *tonal* je desetinné číslo, které nastavuje sílu filtrování histogramu. *infile* a *outfile* jsou cesty k původnímu obrázku, resp. obrázku, kam se má uložit výsledek. Vstupní soubor může být v jakémkoliv formátu, který je na vašem systému schopen otevřít *Imlib2*, výstupní soubor se ukládá ve formátu PNG.

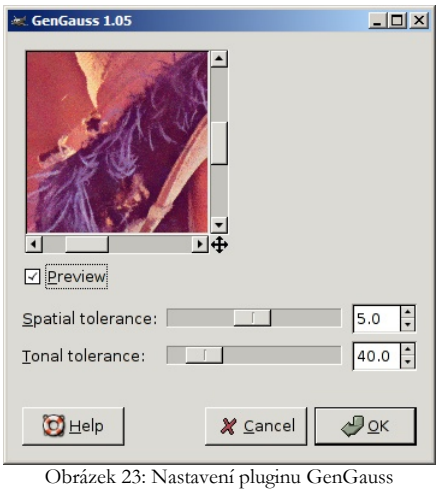
6.1.3. Generalizovaný prostorově – tonální gaussovský filtr – plugin do GIMPu

Systémové požadavky: funkční toolchain, developerský balíček programu GIMP [9] s nástrojem *gimptool-2.0*.

Instalace: plugin je distribuován jako jediný zdrojový soubor *gaussnr*.c a instaluje se pomocí nástroje *gimptool*.

```
gimptool-2.0 --install gaussnr.c
```

Ovládání programu: po instalaci přibude v menu oken GIMPu s otevřeným obrázkem položka *Filters* ► *Enhance* ► *Advanced noise reduction*... Po vyvolání této položky se zobrazí dialog s možnostmi nastavení filtru vyobrazený na obrázku 23. *Spatial tolerance* nastavuje σ gaussiánu použitého pro prostorové filtrování a *Tonal tolerance* nastavuje σ gaussiánu použitého pro tonální filtrování. Podrobnější význam těchto parametrů je rozebrán v kapitole 3.3. Obrázek v horní polovině zobrazuje náhled filtrovaného obrázku se zvolenými parametry. Tlačítkem *OK* se potvrdí volby a spustí filtrování, tlačítkem *Cancel* se zavře okno bez aplikování filtru.



Obrázek 23: Nastavení pluginu GenGauss

6.1.4. Generalizovaný prostorově – tonální gaussovský filtr – konzolová verze

Systémové požadavky: funkční toolchain, developerský balíček knihovny *Imlib2* [10].

Instalace: konzolová verze filtru je distribuována jako zdrojový balíček *gengauss.tar.bz2*. Rozbalení a překlad probíhá následujícím způsobem:

```
tar -xf gengauss.tar.bz2
cd gengauss/Release
make
```

Po skončení překladu je v adresáři *Release* spustitelný soubor *gengauss*, který lze případně, pokud máte administrátorská práva, zkopírovat na vhodné místo v systému (například */usr/local/bin*).

Ovládání programu: program se volá následujícím způsobem:

```
gengauss -s spatial -t tonal infile outfile
```

Všechny parametry jsou povinné. *spatial* nastavuje σ prostorového gaussiánu, *tonal* nastavuje σ tonálního gaussiánu. *infile* je cesta k vstupnímu souboru, *outfile* cesta, kam se má

uložit výsledek. Vstupní soubor může být v jakémkoliv formátu, který je na vašem systému schopen otevřít *Imlib2*, výstupní soubor se ukládá ve formátu PNG.

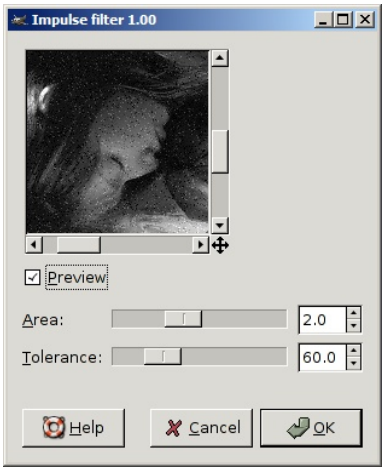
6.1.5. Impulsní filtr – plugin do GIMPu

Systémové požadavky: funkční toolchain, developerský balíček programu GIMP [9] s nástrojem *gimptool-2.0*.

Instalace: plugin je distribuován jako jediný zdrojový soubor *impulse.c* a instaluje se klasicky pomocí nástroje *gimptool*.

```
gimptool-2.0 --install impulse.c
```

Ovládání programu: po instalaci přibude v menu oken GIMPu s otevřeným obrázkem položka *Filters* ►*Enhance* ► *Impulse noise reduction...* Po jejím vyvolání se zobrazí dialog s možnostmi nastavení filtru vyobrazený na obrázku 24. V dialogu je možné nastavit dva parametry: *Area*, který určuje velikost okolí, ze kterého se počítá medián a *Tolerance*, kterým se nastavuje prahová hodnota. Podrobnější význam těchto parametrů je rozebrán v kapitole 4. V horní polovině okna je náhled na rekonstruovaný snímek. Tlačítkem *OK* se potvrzuje nastavení filtru a spouští samotný filtrovací proces. Tlačítkem *Cancel* se zavírá okno bez aplikování filtru.



Obrázek 24: Nastavení impulsního filtru

6.1.6. Impulsní filtr – konzolová verze

Systémové požadavky: funkční toolchain, developerský balíček knihovny *Imlib2* [10].

Instalace: probíhá stejně jako v předchozích případech. Program je distribuován ve zdrojovém balíčku *impulse-standalone.tar.bz2*. Rozbalení a překlad probíhají následovně:

```
tar -xf impulse-standalone.tar.bz2
cd impulse-standalone/Release
make
```

výsledkem překladu je spustitelný binární soubor *impulse*.

Ovládání programu: program se volá následovně:

```
impulse -a area -t tolerance infile outfile
```

kde *area* je velikost okolí, ze kterého se počítá medián a *tolerance* je prahová hodnota, nad kterou považuje program pixel za poškozený. *infile* a *outfile* jsou cesty k vstupnímu, resp. výstupnímu souboru. Vstupní soubor může být v libovolném formátu, který na vašem systému umí přečíst *Imlib2*, výstupní soubor se ukládá ve formátu PNG.

6.1.7. Hot pixels removal tool

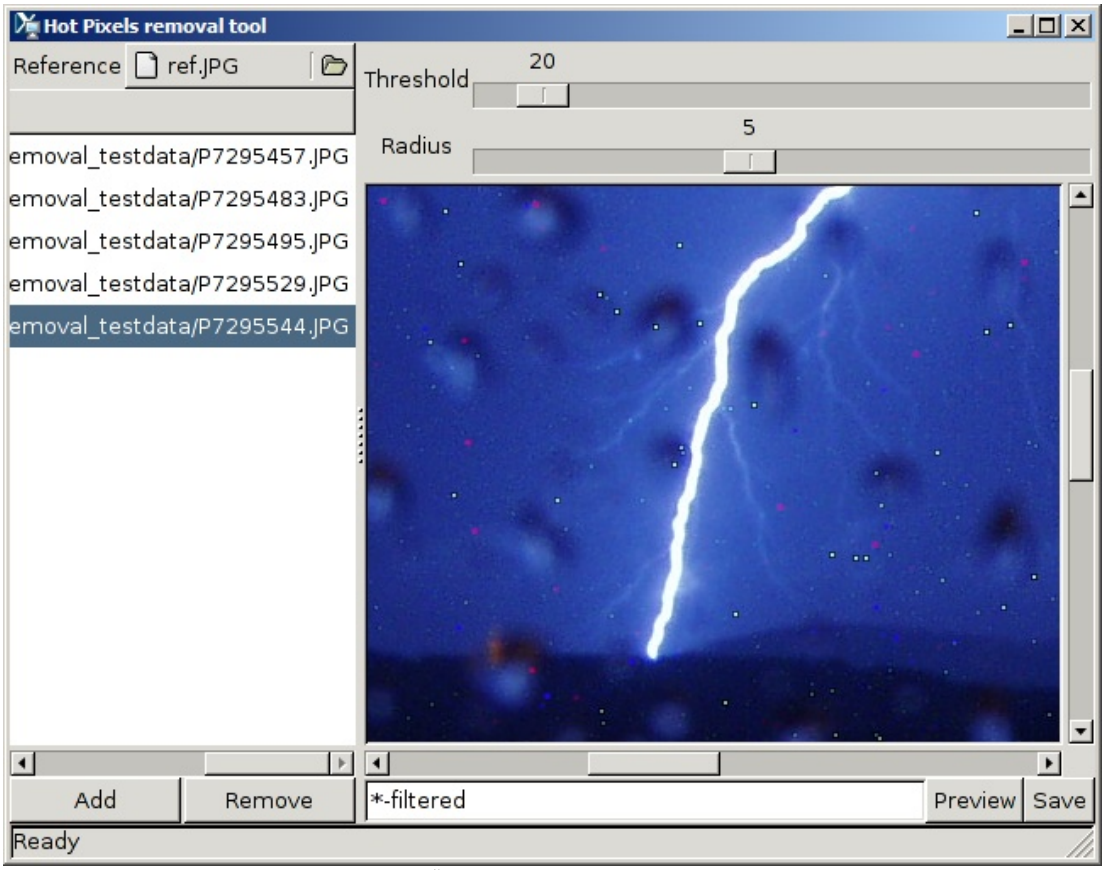
Hot pixels removal tool je implementace postupu popsaného v kapitole 5. Na rozdíl od ostatních programů je implementován jako samostatná grafická aplikace.

Systémové požadavky: funkční toolchain, developerský balíček knihovny *Imlib2* [10], developerský balíček knihovny *gtkmm* [11].

Instalace: Hot pixels removal tool je distribuován jako zdrojový balíček *hotpixel-removal.tar.bz2*. Rozbalení a překlad probíhá následovně:

```
tar -xf hotpixel-removal.tar.bz2
cd hotpixel-removal/Release
make
```

po skončení překladu se v adresáři *Release* objeví spustitelný binární soubor *hotpixel-removal*, který případně, pokud máte administrátorská práva, můžete pomocí programu *install* překopírovat na vhodné místo (*/usr/local/bin*).



Obrázek 25: Uživatelské rozhraní programu *hotpixel-removal*

Ovládání programu: *hotpixel-removal* je navržený pro hromadné zpracování více snímků najednou.

Prvním krokem je načtení referenčního snímku (snímku vyfoceného se zatemněným objektivem). To se provádí kliknutím na tlačítko vedle nápisu *Reference*. Po kliknutí se zobrazí dialogové okno pro výběr souboru. Po potvrzení výběru se referenční snímek pro kontrolu zobrazí v náhledové oblasti.

Následuje načtení zdrojových obrázků. Po kliknutí na tlačítko *Add* se zobrazí dialogové okno pro výběr jednoho nebo více souborů. Po potvrzení jsou tyto soubory přidány do seznamu (označeného *Source files*). Tlačítko *Remove* odstraní ze seznamu právě vybrané položky. Pravá horní část obsahuje ovládací prvky pro nastavení parametrů filtrování: *Threshold* nastavuje prahovou hodnotu filtru a *Radius* rozmazání – to se aplikuje na původní obrázek a z rozmazaného snímku se pak používají pixely pro rekonstrukci.

Po vybrání jedné položky ze seznamu zdrojů se tento snímek zobrazí v náhledové oblasti. Po kliknutí na tlačítko *Preview* se spočítá výsledný snímek a zobrazí se v náhledové oblasti.

Po kliknutí na tlačítko *Save* se přefiltrují a uloží *všechny* snímky vybrané v seznamu zdrojů. Jméno výsledného souboru se nastavuje pomocí kolonky nalevo od tlačítka *Preview* a určuje se následovně:

- Pokud je v kolonce uveden adresář, použije se ten. Jinak se použije adresář, ve kterém leží původní snímek.
- Pokud je v kolonce uvedena přípona, použije se ta. Jinak je použita původní přípona zdrojového snímku.
- Hvězdička v kolonce je nahrazena původním jménem souboru (bez cesty a přípony).

6.2. Programátorská dokumentace

6.2.1. MATLAB

V první fázi práce na tomto projektu byly některé algoritmy implementovány jako skripty v MATLABu. Tyto soubory jsou také na přiloženém CD. Jak je pro MATLAB typické, každá funkce je implementována zvlášť v jednom souboru, jehož jméno se shoduje se jménem funkce, s příponou .m.

`BinSequence(n)`

Vrací vektor s *n*-tým řádkem pascalova trojúhelníku.

`bwimage(Data)`

Zobrazuje černobílé snímky s klasickou grayscale paletou.

`Convolve(what, kernel)`

Provádí konvoluci vektoru *what* vektorem *kernel*. Vrací pouze vektor z prostřední částí výsledku konvoluce, jehož velikost je shodná s velikostí vektoru *what*.

`GaussColor(Image, spatial, tonal)`

Filtruje obrázek *Image* filtrem popsaným v kapitole 3.3. *spatial* a *tonal* jsou parametry filtru. *Image* je barevný obrázek. *GaussColor* počítá tonální vzdálenost ze všech barevných složek najednou.

`GaussKernel(sigma)`

Vrací čtvercovou matici s gaussovským jádrem s rozptylem σ^2.

`GenGauss(Image, spatial, tonal)`

Zpracovává černobílý snímek *Image* algoritmem popsaným v kapitole 3.3. *spatial* a *tonal* jsou parametry filtru.

`GenGaussColor(Image, spatial, tonal)`

Zpracovává barevný snímek *Image*, každou barevnou složku zvlášť (pomocí volání *GenGauss*).

`histogram(Data)`

Vrací vektor délky 256 s histogramem obrázku *Data*.

`ImAdjacent(Data, x, y, sx, sy)`

Vrací okolí bodu *[x, y]* v černobílém obrázku *Data*. Okolí má velikost *[2*sx+1, 2*sy+1]*.

`ImpulseFilter(Image, spatial, threshold)`

Aplikuje na černobílý obrázek *Image* filtr popsaný v kapitole 4. *spatial* a *threshold* jsou parametry filtru.

`LocalMode(histogram, x)`

Nalezne lokální modus k bodu *x* v histogramu *histogram*.

`LocalModeFilter(Image, spatial, tonal)`

Aplikuje na černobílý obrázek *Image* filtr popsaný v kapitole 3.2. *spatial* a *tonal* jsou parametry filtru.

`LocalModeFilterColor(Image, spatial, tonal)`

Aplikuje filtr z kapitoly 3.2. na barevný snímek *Image*. Barevné kanály zpracovává odděleně (pro každý zvlášť volá *LocalModeFilter*).

`normdist(x, sigma)`

Vrací hodnotu funkce hustoty gaussovského rozdělení pravděpodobnosti se střední hodnotou 0 a rozptylem σ^2 v bodě *x*.

`PadImage(Image, pad)`

Rozšíří barevný obraz *Image* na okrajích o *pad* pixelů, přičemž použije zrcadlená data z okrajů.

`rmse(A, B)`

Spočítá střední kvadratickou chybu barevného obrázku *B* proti obrázku *A*.

`rmsebw(A, B)`

Jako *rmse*, ale pro černobílé obrázky.

```
TonalKernel(Image, x, y, s, sigma)
```

Spočítá tonální jádro okolí bodu *[x, y]* v obrázku *Image*. *s* je velikost okolí a *sigma* je parametr σ_t z algoritmu z kapitoly 3.3.

```
WeightedHistogram(Data, kernel)
```

Spočítá vážený histogram z černobílého obrazu *Data* s váhami *kernel*. *Data* a *kernel* musí být matice stejných rozměrů.

6.2.2. localmode-gimp a localmode-standalone

Plugin pro filtrování lokálním modem v GIMPu a konzolová verze tohoto algoritmu sdílejí velkou část kódu. Obě dvě verze jsou implementovány pouze v jednom souboru (který se v obou případech jmenuje *localmode.c*). Následuje popis jednotlivých funkcí (které mají v obou případech stejnou funkci):

```
static void query(void)
```

je registrační funkce GIMPovských pluginů.

```
static void run (const gchar *name, gint nparams,
const GimpParam *param, gint *nreturn_vals, GimpParam
**return_vals)
```

je hlavní funkce GIMPovských pluginů. Význam parametrů viz programátorská dokumentace ke GIMPu [9].

```
static void localmode(GimpDrawable *drawable, gdouble
spatial, gdouble tonal)
```

je funkce, která pouze volá *localmode_region* pro celý obrázek. *drawable* je obrazová struktura GIMPu se zpracovávaným obrazem, *spatial* a *tonal* jsou parametry filtru.

```
gdouble norm_dist(double x, double sigma)
```

je funkce, která vrací gaussián v bodě *x* při rozptylu σ^2.

```
gdouble *gauss_kernel(const gdouble sigma)
```

je funkce, která naalokuje a vrátí pole obsahující 2D gaussián s rozptylem σ^2.

```
static void localmode_region(GimpPixelRgn *srcPR,
GimpPixelRgn *destPR, gint bytes, gdouble spatial,
gdouble tonal, gint x1, gint x2, gint y1, gint y2,
gboolean show_progress)
```

je funkce se samotnou implementací algoritmu z kapitoly 3.2. *srcPR* a *destPR* jsou zdrojové a cílové regiony v obraze, *bytes* je počet bytů na pixel, *spatial* a *tonal* jsou parametry filtru, *x1, x2, y1, y2* jsou koordináty udávající velikost a pozici zpracovávaného okénka a

show_progress je příznak, při jehož nastavení posílá funkce GIMPu zprávy pro překreslení progressbaru.

```
static gboolean localmode_dialog(GimpDrawable
*drawable)
```

je funkce, která vykresluje dialog pro nastavení parametrů. *drawable* je parametr s obrazovou strukturou GIMPu pro náhled.

```
static void preview_update(GimpPreview *preview)
```

je funkce, která přepočítá náhled (zavoláním *localmode_region*).

6.2.3. gengauss-gimp a gengauss-standalone

Stejně jako v předchozím případě, *gengauss-gimp* a *gengauss-standalone* sdílejí velkou část kódu a oba dva jsou implementovány jedním souborem (který ve pro GIMPovskou verzi jmenuje *gaussnr.c* a pro standalone verzi *gengauss.c*). Následuje popis jednotlivých funkcí (které mají v obou případech stejnou funkci):

```
static void query(void)
```

je registrační funkce GIMPovských pluginů.

```
static void run (const gchar *name, gint nparams,
const GimpParam *param, gint *nreturn_vals, GimpParam
**return_vals)
```

je hlavní funkce GIMPovských pluginů. Význam parametrů viz programátorská dokumentace ke GIMPu [9].

```
static void gen_gauss(GimpDrawable *drawable, gdouble
spatial, gdouble tonal)
```

je funkce, která pouze volá *gen_gauss_region* pro celý obrázek. *drawable* je obrazová struktura GIMPu se zpracovávaným obrazem, *spatial* a *tonal* jsou parametry filtru.

```
gdouble norm_dist(double x, double sigma)
```

je funkce, která vrací gaussián v bodě x při rozptylu σ^2.

```
gdouble *gauss_kernel(const gdouble sigma)
```

je funkce, která naalokuje a vrátí pole obsahující 2D gaussián s rozptylem σ^2.

```
static void gen_gauss_region(GimpPixelRgn *srcPR,
GimpPixelRgn *destPR, gint bytes, gdouble spatial,
gdouble tonal, gint x1, gint x2, gint y1, gint y2,
gboolean show_progress)
```

je funkce se samotnou implementací algoritmu z kapitoly 3.3. *srcPR* a *destPR* jsou zdrojové a cílové regiony v obraze, *bytes* je počet bytů na pixel, *spatial* a *tonal* jsou parametry filtru, *x1, x2, y1, y2* jsou koordináty udávající velikost a pozici zpracovávaného okénka a *show_progress* je příznak, při jehož nastavení posílá funkce GIMPu zprávy pro překreslení progressbaru.

```
static gboolean gen_gauss_dialog(GimpDrawable
*drawable)
```

je funkce, která vykresluje dialog pro nastavení parametrů. *drawable* je parametr s obrazovou strukturou GIMPu pro náhled.

```
static void preview_update (GimpPreview *preview)
```

je funkce, která přepočítá náhled (zavoláním *gen_gauss_region*).

6.2.4. impulse-gimp a impulse-standalone

Obě implementace opět sdílí velkou část kódu a jsou celé implementované v jednom zdrojovém souboru (který se v obou případech jmenuje *impulse.c*). Následuje popis jednotlivých funkcí:

```
static void query(void)
```

je registrační funkce GIMPovských pluginů.

```
static void run (const gchar *name, gint nparams,
const GimpParam *param, gint *nreturn_vals, GimpParam
**return_vals)
```

je hlavní funkce GIMPovských pluginů. Význam parametrů viz programátorská dokumentace ke GIMPu [9].

```
static void impulse(GimpDrawable *drawable, gdouble
spatial, gdouble tonal)
```

je funkce, která pouze volá *impulse_region* pro celý obrázek. *drawable* je obrazová struktura GIMPu se zpracovávaným obrazem, *spatial* a *tonal* jsou parametry filtru.

```
static void impulse_region(GimpPixelRgn *srcPR,
GimpPixelRgn *destPR, gint bytes, gint area, gint
tolerance, gint x1, gint x2, gint y1, gint y2,
gboolean show_progress)
```

je funkce se samotnou implementací algoritmu z kapitoly 4. *srcPR* a *destPR* jsou zdrojové a cílové regiony v obraze, *bytes* je počet bytů na pixel, *area* a *tolerance* jsou parametry filtru, *x1, x2, y1, y2* jsou koordináty udávající velikost a pozici zpracovávaného okénka a *show_progress* je příznak, při jehož nastavení posílá funkce GIMPu zprávy pro překreslení progressbaru.

```
static gboolean impulse_dialog(GimpDrawable *drawable)
```

je funkce, která vykresluje dialog pro nastavení parametrů. *drawable* je parametr s obrazovou strukturou GIMPu pro náhled.

```
static void preview_update(GimpPreview *preview)
```

je funkce, která přepočítá náhled (zavoláním *impulse_region*).

6.2.5. hotpixel-removal

hotpixel-removal je program napsaný v C++ s použitím knihovny *gtkmm* [11]. Zdrojáky sestávají z pěti souborů: *engine.cpp, engine.h, interface.cc, interface.h, main.cc*. Následuje popis důležitých tříd:

Interface

```
class AppWindow:public Gtk::Window{
     public:
     AppWindow();
     virtual ~AppWindow();
};
```

widget s hlavním oknem aplikace

```
class ReferenceSelector:public Gtk::HBox{
     public:
     ReferenceSelector();
     virtual ~ReferenceSelector();
     string GetFilename(void);
     sigc::signal<void, Request*> signal_request;
     };
```

widget pro vybírání referenčního snímku. *GetFilename* vrací cestu k momentálně vybranému souboru. Pokud je referenční snímek změněn, emituje se signál *signal_request* s *ShowRequest* na zobrazení referenčního snímku.

```
class SourceSelector:public Gtk::VBox{
     public:
     SourceSelector();
     virtual ~SourceSelector();
     list<string> *GetSelectedItems(void);
     sigc::signal<void, Request*> signal_request;
     };
```

widget se seznamem snímků a tlačítky *Add* a *Remove*. *GetSelectedItems* vrací list s vybranými položkami. Když uživatel označí v seznamu jeden snímek, emituje se *signal_request* s požadavkem *ShowRequest* na zobrazení vybraného snímku.

```
class Settings:public Gtk::Table{
     public:
     Settings();
     int GetThreshold(void);
     int GetRadius(void);
     };
```

widget s hejbátky pro nastavování hodnot *Threshold* a *Radius*. Funkce *GetThreshold* a *GetRadius* slouží k získání nastavených hodnot.

Engine

```
class WorkerThread{
     public:
     Glib::Dispatcher ResponseDispatcher;
     WorkerThread();
     ~WorkerThread();
     Request *GetResults(void);
     void on_request(Request *req);
     };
```

Objekt třídy WorkerThread okamžitě po vytvoření spustí další vlákno, ve kterém čeká na požadavky a zpracovává je. O zpracování požadavku (třídy *Request*) si požádáme metodou *on_request*. Poté, co bude požadavek vyřízen, emituje *ResponseDispatcher* signál a handler na něj napojený pak může pomocí metody *GetResults* vyzvednout vyřízený požadavek.

```
class Request{
     public:
     int Width, Height;
     uint8_t *PData;
     string Message;
     virtual void work(void)=0;
     virtual ~Request();
     protected:
     Request(void);
     };
```

Request je abstraktní předek všech požadavků. Jméno možná nebylo zvoleno vhodně, protože se v něm ukládá takříkajíc i response. Pokud je výsledkem našeho požadavku nějaký obrázek, který se má zobrazit na náhledová ploše, uloží se obrazová data do *PData* a rozměry do *Width* a *Height*. Pokud si žádný obrázek zobrazit nepřejeme, nastavíme *PData=NULL*. Pokud si přejeme po vyřízení requestu zobrazit ve stavovém řádku nějakou zprávu, uložíme jí do *Message*. *work* je virtuální metoda, která je implementovaná v potomcích *Request* a kterou *WorkerThread* volá ve vlákně běžícím na pozadí.

Následující třídy jsou potomci *Request*. Liší se pouze rozdílnými implementacemi metody *work*.

```
class ShowRequest:public Request
```

požadavek na načtení obrázku a zobrazení na náhledové ploše

```
class PreviewRequest:public AbstractProcessRequest
```

požadavek na přefiltrování obrázku a zobrazení výsledku na náhledové ploše

```
class SaveRequest:public AbstractProcessRequest
```

požadavek na načtení jednoho nebo více obrázků, jejich přefiltrování a uložení na disk.

7. Obsah CD

Přiložené CD obsahuje zdrojové soubory s implementacemi algoritmů, elektronickou verzi textu bakalářské práce, několik testovacích obrázků několik dalších výsledků popsaných filtrovacích algoritmů. Následuje obsah jednotlivých adresářů:

text - elektronická podoba textu bakalářské práce

results - soubory se ukázkovými výsledky filtrování

gengauss-gimp - zdrojový soubor implementace algoritmu GenGauss jako pluginu do GIMPu.

gengauss-standalone - konzolová verze GenGauss.

hotpixel-removal - zdrojový balíček programu hotpixel-removal a několik testovacích fotek

impulse-gimp – zdrojový soubor implementace impulsního filtru jako pluginu do GIMPu.

impulse-standalone – konzolová verze impulsního filtru

localmode-gimp - zdrojový soubor implementace LocalMode filtru jako pluginu do GIMPu.

localmode-standalone – konzolová verze filtru LocalMode

8. Závěr a zhodnocení

Výsledkem této práce je teoretické i praktické prozkoumání a implementace celkem čtyřech různých algoritmů. Tři z nich se ukázaly jako dobře použitelné; obzvlášťe algoritmus *GenGauss* produkuje velmi dobré výsledky. Jejich implementace poskytuje sadu nástrojů, která dokáže účinně opravit velké množství obrazových vad digitálních fotografií.

9. Literatura a odkazy

[1] S. Peng and L. Lucke (1995). A Hybrid Filter for Image Enhancement. *Proceedings of the 1995 International Conference on Image Processing (ICIP'95)*:163-166.

[2] D.R.K. Brownrigg (1984). The weighted median filter. *Commun. ACM*, 27:807-818.

[3] L.S. Davis and A. Rosenfeld (1987). Noise cleaning by iterated local averaging. *IEEE Trans. on Systems, Man and Cybernetics*, SMC-8(9):705-710.

[4] PictureCode: Noise Ninja, *http://www.picturecode.com/*

[5] ABSoft: Neat Image, *http://www.neatimage.com/*

[6] J. van de Weijer and R. van den Boomgaard (2001). Local Mode Filtering. *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'01).*

[7] R. A. Boie and I. J. Cox (1992). An Analysis of Camera Noise. *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 14, no. 6:* 671-674.

[8] I. Pitas, A. N. Venetanopoulos (1989). Nonlinear Digital Filters. *Kluwer.*

[9] The GNU Image Manipulation Program, *http://www.gimp.org/*

[10] Enlightenment.org: Imlib2, *http://enlightenment.sourceforge.net/Libraries/Imlib2/*

[11] gtkmm, *http://www.gtkmm.org/*