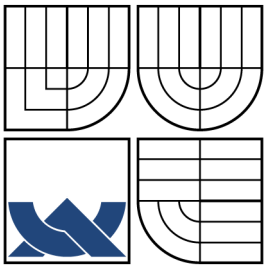
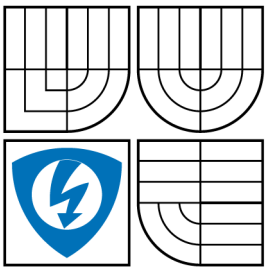


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND
COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

OBRAZOVÉ FILTRY PRO EVOLUČNÍ PROGRAMOVÁNÍ

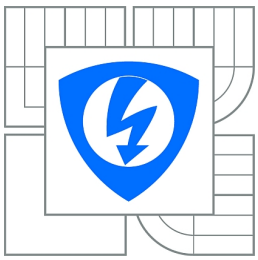
IMAGE FILTERS FOR EVOLUTIONARY PROGRAMMING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — BC. MILOŠ ZAVADIL
AUTHOR

VEDOUCÍ PRÁCE — ING. RADIM BURGET
SUPERVISOR

BRNO 2009



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Miloš Zavadil	***ID:***	78348
Ročník:	2	***Akademický rok:***	2009/2010

NÁZEV TÉMATU:

Obrazové filtry pro evoluční programování

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s dostupnými knihovnami pro zpracování obrazu (napr(. JImage) a vzájemne(je srovnejte. Popište principy genetického programování a navrhne(te způsob, jak lze spolu s ním navrhovat obrazové filtry. Popište strukturu zvolené knihovny a vytvor(te nejméně 5 různých komponent pro genetické programování v jazyce JAVA.

DOPORUČENÁ LITERATURA:

[1] LYON D., A., Image Processing in Java, Prentice Hall PTR, 1999
[2] BURGER W., BURGE M., J., Digital Image Processing: An Algorithmic Introduction using Java, Springer; 1 edition (November 28, 2007)

Termín zadání:	29.1.2010	***Termín odevzdání:***	26.5.2010

Vedoucí práce: Ing. Radim Burget

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

Obrazové filtry představují část vědní disciplíny zabívající se digitálním zpracováním obrazu. Filtrace obrazu slouží ke zvýraznění určité informace. Můžeme potlačit šum, vyhladit obraz, zvýraznit kontrast, nebo detekovat hrany. Samotný návrh obrazových filtrů představuje časově poměrně náročný proces. Je tedy žádoucí jej v maximální míře zautomatizovat a přenechat činnost předem naprogramovanému systému. Práce se zabývá návrhem komponent pro zmíněný systém. Jedná se o část funkce expertního systému, kterému se na vstupu poskytne množina vstupních komponent, z nichž se za použití principů evolučního programování mohou generovat nové filtry a následně vyhodnocovat jejich spolehlivost pro další využití.

KLÍČOVÁ SLOVA

Obrazové filtry, Evoluční programování, knihovna JImage

ABSTRACT

Image filters is a subset of signal processing. Image filtering is mainly used for highlighting an information. It can be useful for reduce noise, smooth pictures, enhance contrast or for edge detection. Image filter design itself is a time-consuming process. It is suitable to automate the process and give up the function of filter designing to preprogrammed system. Designing komponents for that system is aim of this work. It is part of an whole expert system. A set of information is given on input which are used for generating new image filters. Subsequently it will evaluate the relevancy of concrete image filter for subsequent use.

KEYWORDS

Image Filters, Evolutionary programming, JImage Library

ZAVADIL, M. *Obrazové filtry pro evoluční proramování. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2010. 57 s.Vedoucí diplomové práce Ing. Radim Burget.*

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Obrazové filtry pro evoluční programování“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 152 trestního zákona č. 140/1961 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Radimovi Burgetovi za velmi užitečnou metodickou pomoc a cenné rady při zpracování diplomové práce.

V Brně dne ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Diplomová práce se zabývá obrazovými filtry pro evoluční programování. Jedná se o část evolučního expertního systému, který bude sloužit k výzkumu nových obrazových filtrů, s využitím evolučních technik. Prvotní využití předpokládá nasazení pro diagnostiku zdravotnických dat a automatickou detekci patologií ve zdravotnických datech. Jedná se o data z různých zdravotnických zařízení s rozličnou kvalitou obrazu. Ta nemusí být vždy ideální a pro rozeznání důležitých detailů, či nalezení potřebných patologií je třeba obraz patřičně upravit. K tomuto účelu slouží obrazové filtry. Ty mohou důležité části obrazu zvýraznit, nepotřebné naopak potlačit. Můžeme také odstranit nechtěný šum nebo naopak zvýraznit hrany obsažených objektů. Obecně jsme schpni na jeden obraz aplikovat libovolné množství operací. Máme tedy v rukou poměrně mocný nástroj jak upravovat obrazová data, ale nemáme systém, který by nám řekl, jaké úpravy použít na konkrétní problém, aby se provedené změny nestaly ve výsledku kontraproduktivními.
K tomuto účelu by měl sloužit systém jako celek, kdy expertnímu systému poskytneme množinu vstupních komponent. Ten s využitím principů evolučního programování bude generovat nové filtry a vyhodnocovat jejich spolehlivost. To by se mělo dít na základě vytvořené populace obsahující velké množství jedinců (obrazových filtrů), které se mezi sebou budou křížit a budou mutovat. Dle posouzení kvality jednotlivých jedinců budou ve většině případů postupovat v evoluci jedinci s nejlepšími výsledky.
Samotná práce se tématicky dělí do pěti kapitol. V první je čtenář teoreticky seznámen s genetickým programováním a jsou dány v souvislost jeho základní pojmy. Druhá část čtenáři poskytuje vstupní znalosti o obarzových filterch, jež jsou dále v práci využity pro samotný návrh obrazových filtrů. Ve třetí části jsou diskutovány dostupné knihovny pro zpracování obrazu, jsou vzájemně porovnány a jsou uvedeny jejich výhody a nevýhody pro případné praktické použití. Ve čtvrté části je popsána knihovna JImage z pohledu instalace, uživatelského využití, je popsána její vnitřní struktura z programátorského hlediska a jsou diskutovány její možnosti pro zpracování obrazu. Výše zmíněné představuje nezbytné vstupní vědomosti, bez nichž by nebylo možné realizovat poslední část práce. Ta prezentuje návrh jednotlivých komponent pro GP, jsou uvedeny jejich případné nedostatky a řešení jak je odstranit. Jsou prezentovány dosažené výsledky a v závěru je diskutováno začlenení vytvořených komponent do evolučního systému.

1 GENETICKÉ PROGRAMOVÁNÍ

Základním pojmem genetického programování je genetický algoritmus. Jedná se o heuristický postup snažící se pomocí principů evoluční biologie nalézt řešení složitých problémů, pro které je obtížné nalézt, či stanovit přesný algoritmus. [2] Genetické algoritmy používají techniky napodobující přirozené evoluční procesy známé z biologie, tj. dědičnost, mutace, přirozený výběr, křížení [1][3].
V obecné rovině lze konstatovat, že princip práce genetického algoritmu je postupná tvorba generací různých řešení daného problému. Při jeho řešení se pak uchovává tzv. populace, jejíž každý jedinec představuje jedno řešení daného problému. Jak populace probíhá evolucí, řešení daného problému se zlepšuje. Tradičně je řešení reprezentováno binárnímy čísly, nicméně používají se i jiné reprezentace, např. strom, pole, matice, apod. Ve většině případů je na začátku simulace (v první generaci) populace složena z naprosto náhodných členů. V přechodu do nové generace je pro každého jedince spočtena tzv. fitness funkce, jež vyjadřuje kvalitu řešení reprezentovaného tímto jedincem. Dle uvedeného kritéria jsou stochasticky vybráni jedinci, kteří jsou dále modifokováni (pomocí mutací a křížení), čímž vznikne nová populace. Uvedený postup se iterativně opakuje, čímž se kvalita řešení v populaci postupem času vylepšuje. Algoritmus se většinou ukončuje po splnění předem dané podmínky postačující kvality řešení, případně po předem dané době [1].

Popis genetického algoritmu

1. **(Inicializace)** Vytvoř nultou populaci (obvykle složenou z náhodně vygenerovaných jedinců)

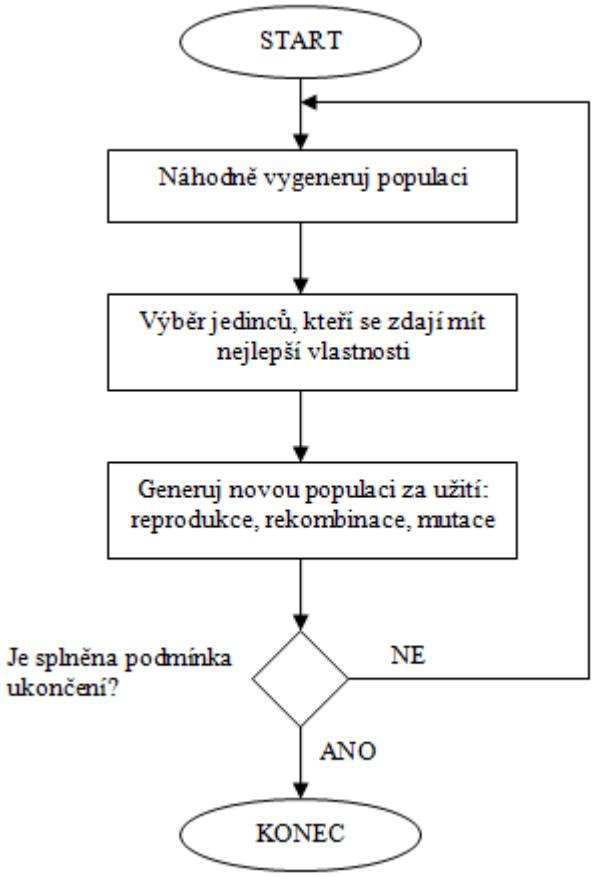
2. **(Začátek cyklu)** Pomocí určité výběrové metody (zpravidla zčástí náhodné) vyber z populace několik jedinců s vysokou zdatností

3. Z vybraných jedinců vygeneruj nové použitím metod reprodukce, rekombinace, mutace.

4. Dle výpočtu zdatnosti se celý algoritmus vrací k bodu 2 v případě nesplněné podmínky zastavovací funkce, naopak je-li splněna zastavovací podmínka, algorimus je ukončen.

5. **(Konec algoritmu)** Jedinec s nejvyšší zdatností je hlavním výstupem algoritmu a reprezentuje nejlepší nalezené řešení.

Výše popsaný algoritmus je graficky znázorněn na obrázku 1.1



Obrázek 1.1: Obecný genetický algoritmus

1.1 Základní pojmy

Pro jedince se používá označení fenotyp a pro jeho reprezentaci se používá termín genotyp, genom nebo chromozom. Chromozom se dělí na jednotlivé geny, které jsou uspořádány. To znamená, že i-tý gen chromozomů stejného typu reprezentuje stejnou charakteristiku. Gen nabývá různých hodnot, kterým se říká alely. Jedinci mohou být zakódováni (geneticky popsáni) různými způsoby. To, jakým způsobem jsou popsáni, může být důležité pro úspěch či neúspěch řešení konkrétní úlohy. Jedním z jednoduchých způsobů může být například binární řetězec dané délky [1][2].

1.2 Mutace

Principielně zavádí mutace do chromozomu jistou chybu. Díky ní je možno překonat lokální optima a hledat i v jiných oblastech. Můžeme tedy konstatovat, že mutace jsou vhodné zejména pro ten případ, kdy konvergujeme k nějakému řešení, jež uvázlo v lokálním optimu a právě tohle optimum potřebujeme překonat za účelem nalezení

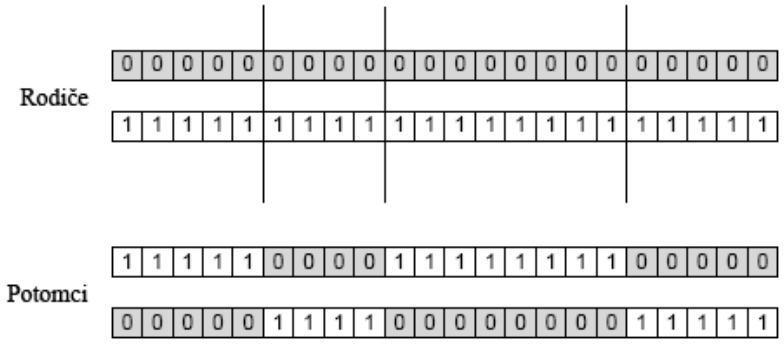
globálního optima. Nevýhodou je, že mutace jako taková většinou nevede ke zlepšení a proto se využívá s menší pravděpodobností, aby zbytečně neměla vliv na zdokonalování současné populace.

1.3 Křížení

Při operaci křížení dochází ke kombinaci dvou jedinců (reprezentovanými chromozomem).

1.3.1 N - bodové křížení

V případě N - bodového křížení je genom rozdělen na několik nerovnoměrných částí, u nich pak dochází ke změnám. Celá situace je nejlépe zřejmá z níže uvedeného obrázku (Obr. 1.2).



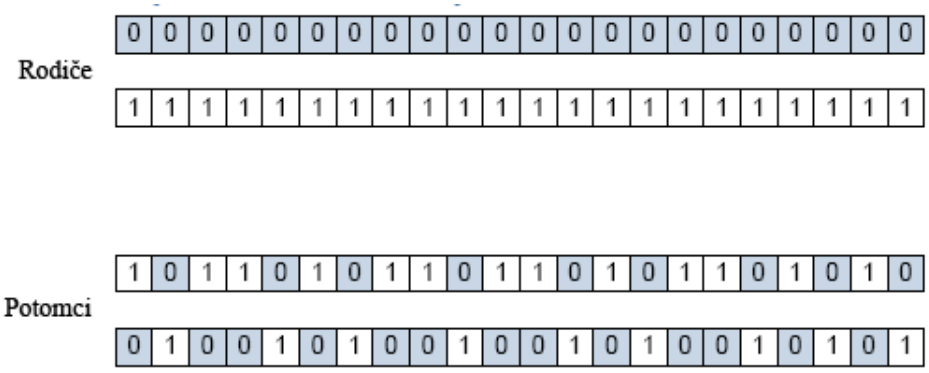
Obrázek 1.2: N - bodové křížení

1.3.2 Rovnoměrné křížení

V případě rovnoměrného křížení je genom rozdělen na několik rovnoměrných částí, u nich pak dochází ke změnám (Obr. 1.3).

1.3.3 Křížení vs. mutace

Implementace křížení či mutace závisí na konkrétním typu problému, který má genetický algoritmus řešit. Je třeba mít na paměti fakt, že zatímco křížení může být nasazeno samostatně, pouhá mutace nikdy fungovat nemůže. Obecně se ale považuje za optimální nasazení jak křížení tak mutace současně, neboť každý z nich má jinou roli. Úkolem křížení je posunout řešení blíže směrem k předpokládanému výsledku a kombinuje výhody obou potomků. Naproti tomu mutace vytváří drobné odchylky



Obrázek 1.3: Rovnoměrné křížení

a zanáší do měření náhodnost, resp. novou informaci. Pokud by mutace převážila efekt křížení, eliminovala by tím konvergenci celého systému a algoritmus by tak nebyl schopen nekonvergovat k finálnímu řešení. Na druhé straně bude-li mutace velice malá, popřípadě žádná, hrozí nebezpečí, že algoritmus uvázne v lokálním optimu a nedojde nalezení globálního optima. Nastavení algoritmu je tedy jakýmsi kompromisem mezi oběma výše uvedenými metodami.

1.4 Přirozený výběr

Přirozený výběr je proces, který vybírá ty chromozomy, které mají přežít do další generace. Existuje mnoho variant. Mezi nejběžněji používané patří klasický přirozený výběr, výběr dle nejlepšího chromozomu, post výběr, prahový výběr, soutěživý výběr či výběr způsobem rulety. K tomu, aby mohly být chromozomy vzájemně porovnávány, musí implementovat tzv. fitness funkci. Jedná se o funkci, která je navržena vždy pro každý druh genu a na základě jeho stavby se pokouší o jeho hodnocení a tedy i porovnávání. [1]

1.4.1 Výběr dle nejlepšího chromozomu

Výběr nejlepšího chromozomu probíhá na základě seřazení chromozomů dle hodnoty jejich fitness funkce. Z tohoto seznamu je poté vzato N nejlepších jedinců, kteří jsou vyvoleni pro pokračování v další evoluci.

1.4.2 Prahový výběr

Prahový výběr je realizován stejně jako v předchozím případě, s tím rozdílem, že počet prvků není dán jako konstanta, ale bere se v potaz předem stanovené kritérium, které musí splnit fitness funkce každého chromozomu.

1.4.3 Původní přirozený výběr

V tomto případě je vybráno N nejlepších genů, ovšem není zaručeno, že jejich pořadí bude striktně dodrženo, ba dokonce se může do další evoluce dostat i gen nižší kvality, zatímco gen s vyšší kvalitou vypadne. Majoritně tedy bereme jako směrodatnou kvalitu genu, nicméně zavádí se v této metodě výběru jistá míra náhody. Ta může mít v dalších krocích za následek rychlejší konvergenci k žádanému výsledku.

1.4.4 Post výběr

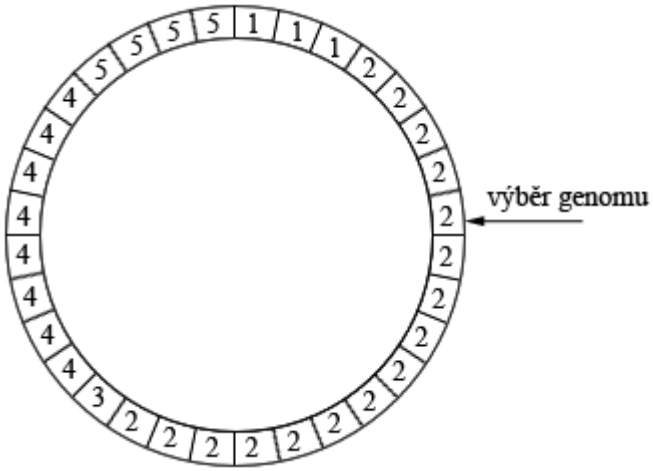
Post výběr porovnává svojí kvalitu až na základě kvality genetického materiálu, který předá svým potomkům. Jinými slovy, na základě genetické kvality potomků rozhodneme o přežití jejich rodičů, potažmo celého společenstva. V praxi se ovšem uvedený druh výběru vyznačuje vyššímy nároky na paměť a výkon počítače.

1.4.5 Soutěživý výběr

Probíhá pořádáním turnajů mezi jednotlivými geny. Ty mají za úkol řešit jistý problém, výběr je následně uskutečněn dle nejlepších výsledků řešení problému.

1.4.6 Výběr pomocí rulety

V první fázi algoritmus umisťuje jednotlivé geny do rulety. To se děje na základě fitness funkce. Dle její hodnoty je danému genomu na ruletě přiřazen patřičný počet míst. Výběr je pak realizován ukazatelem do rulety a vybrán je genom, jež na dané pozici zabírá místo. S rostoucí hodnotou fitness funkce genom zabírá více míst na ruletě a značně tak zyvšuje svoji pravděpodobnost na úspěšný výběr.



Obrázek 1.4: Výběr pomocí rulety z pěti vzorků

Princip je patrný z obrázku 1.4, kde o výběr soutěží 5 genomů. Jejich pravděpodnost výběru je popořadě následující:

- genom č.1 = 9,375%
- genom č.2 = 50%
- genom č.3 = 3,125%
- genom č.4 = 25%
- genom č.5 = 12,5%

1.5 Genetické programování

Genetické programování je součástí skupiny evolučních algoritmů, má tedy společný základ jako GA (genetické algoritmy) s tím rozdílem, že modifikují symboly, které představují program. Tyto symboly mohou mít proměnlivou délku a samozřejmě proměnlivý tvar. V obecnějším smyslu můžeme genetické programování dokonce považovat za způsob strojového učení, tedy vědního oboru, který se zabývá výzkumem algoritmů schopných se učit na základě předchozí zkušenosti. Podobnost je vidět zejména u algoritmů, které stály na počátku této vědy [3].
Genetické programování lze použít k tomu, aby se vypořádal s širokou škálou problémů, počínaje satelitních ovladačů, jako odvětví umělé inteligence, či řešení složitých problémů, které závisí na spoustě parametrů a vzájemně se ovlivňují. Genetické programování je obecně daleko mocnější než genetické algoritmy. Zatímco v případě genetických algoritmů byly výstupem optimální parametry, v případě genetického programování to jsou programy. Je to tedy počátek počítačových programů, které píšou počítačové programy. Uvedené posupy dávají výborné výsledky zejména pro takové problémy, pro které neexistuje ideální řešení. [1]
Můžeme říci, že základní myšlenka vychází z biologických principů. V průběhu miliard let utvořila pozemská evoluce z původních organických látek neuvěřitelné kombinace látek a struktur, jež jsou dnes základem nejrůznějších chemických, či průmyslových odvětví. Pokud by se podařilo nastartovat podobný vývoj i v dnešní softwarové problematice, mohli bychom se v průběhu času dostat ke zcela netušeným výsledkům.

2 DIGITÁLNÍ ZPRACOVÁNÍ OBRAZU - OBRAZOVÉ FILTRY

Na digitální obraz je vhodné nahlížet jako na 2D funkci $f(x, y)$, kde x, y představují prostorové rovinné souřadnice a funkční hodnota funkce f odpovídá jasu, resp. i chromatickému popisu v daném místě [7]. Aby se ovšem daný obraz stal digitálním v pravém slova smyslu, je nezbytné provést úpravy spočívající v diskretizaci souřadnic i jasu. První jmenovaná je realizována pomocí vzorkování, druhá pak kvantováním. Výsledný obraz po vzorkování je sestaven z konečného počtu elementů, z nichž každý má svoji diskrétní polohu a hodnotu. Zmíněné elementy se nazývají jako obrazové elementy neboli pixely. V případě, že jsou funkční hodnoty diskrétní funkce $f(x, y)$ z konečné množiny diskrétních čísel, lze obraz označit za digitální [7][9]. Celý takto provedený proces lze označit jako rasterizace. Takto rastrovaný obraz je rozdělen na konečný počet pixelů a pomocí kvantování je pak jednotlivým pixelům přiřazena hodnota jasu z dané množiny diskrétních hodnot. Přímou souvislost s uvedeným postupem má kvalita obrazu, která je dána především počtem pixelů a počtem kvantovacích hladin jejich jasů.

Jelikož je obraz $f(x, y)$ vzorkován tak, že obsahuje M řádků a N sloupců, je možné pro jejich indexaci použít celá kladná čísla s počátkem o souřadnicích (0,0). Z uvedeného vyplívá možnost matematicky interpretovat výše zmíněné operace jako matici reálných čísel. Jednak ve formě dvourozměrné funkce

$$f(x, y) = \begin{bmatrix} f(0,0) & f(0,1) & \dots & f(0,N-1) \\ f(1,0) & f(1,1) & \dots & f(1,N-1) \\ \vdots & \vdots & \ddots & \vdots \\ f(M-1,0) & f(M-1,1) & \dots & f(M-1,N-1) \end{bmatrix} \tag{2.1}$$

nebo ve formě maticové

$$\mathbf{A} = \begin{bmatrix} a_{0,0} & a_{0,1} & \dots & a_{0,N-1} \\ a_{1,0} & a_{1,1} & \dots & a_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M-1,0} & a_{M-1,1} & \dots & a_{M-1,N-1} \end{bmatrix} \tag{2.2}$$

Jedná o dvě identické matice, kde $a_{i,j} = f(x = i, y = j) = f(i, j)$, lišící se pouze zápisem [7].

2.1 Základy prostorové filtrace

Princip filtrace je realizován posunem jádra po jednotlivých pixelech obrazu, kdy je na každé souřadnici (x, y) určena odezva filtru pomocí definovaného algoritmu.

Budeme-li hovořit o lineární prostorové filtraci je odezva vypočtena jako součet součinů jednotlivých koeficientů filtru a odpovídajících pixelů překrytých filtrační maskou. V případě filtrační masky o velikosti 3x3 je odezva, označená jako R, pro jeden obrazový bod (x, y) vypočtena jako

$$\begin{aligned} R = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + \ldots \\ +w(0,0)f(x,y) + \ldots + w(1,0)f(x+1,y) + w(1,1)f(x+1,y+1). \end{aligned} \tag{2.3}$$

Z výše uvedeného lze vysledovat překryv koeficientu masky $w(0,0)$ s pixelem $f(x,y)$, maska je tedy centrována s bodem (x,y).
Obecný vztah pro výpočet prostorové filtrace obrazu f o velikosti $M \times N$ pomocí filtrační masky w o velikosti $m \times n$ je dán vztahem

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{2.4}$$

kde $a = (m-1)/2$ a $b = (n-1)/2$. Pro filtraci kompletně celého obrazu musí být tento výpočet proveden pro všechna $x = 0, 1, 2, \ldots, M-1$ a $y = 0, 1, 2, \ldots, N-1$, resp. pro každý pixel v obraze musí být spočítána odezva daného filtru.
Zápis výpočtu odezvy filtru velikosti $m \times n$ lze účelně zjednodušit na

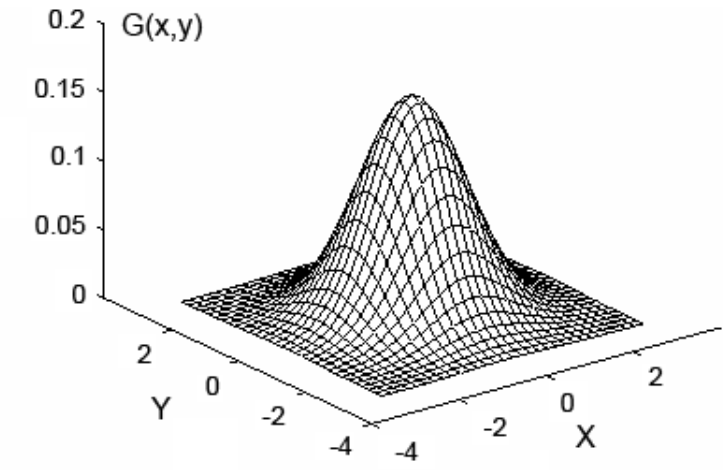
$$R = w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn} = \sum_{i=1}^{mn} w_i zi \tag{2.5}$$

kde přiřazení hodnot koeficientů w_i odopovídá prvkům masky $w(s,t)$ a hodnoty z_i jsou jasové hodnoty obrazové funkce překryté odpovídajícími koeficienty w_i.
Uvedený algoritmus však skýtá úskalí při jeho implementaci. Jedná se o jeho chování u hranic celého obrazu. V případě, kdy budeme chtít zjistit odezvu počátečního bodu obrazu o souřadnicích $(x = 0, y = 0)$ bude část masky mimo obraz. Postupů, jak řešit vzniklou situcí, je víc. Nejjednodušším způsobem pro ošetření pohybu filtrační masky (o velikosti $n \times n$) je zákaz pohybu jejího středu do menší vzdálenosti od kraje než $(n-1)/2$. Všechny tyto pixely ve výsledném obraze nebudou nebo budou mít nulovou hodnotu. Výhodou tohoto postupu je fakt, že všechny pixely ve výsledném obraze jsou vypočteny pomocí kompletní filtrační masky.

2.2 Vyhlazovací prostorové filtry

Jedná se o takové filtry, jejichž odezva se stanoví jako váhovaný průměr pixelů v okolí daném filtrační maskou. V obrazu dojde ke znatelnému potlačení vysokých frekvencí, jež jsou reprezentovány ostrými hranami v obraze. Tyto hrany jsou identifikovatelné jako velké změny jasu na malém prostoru. Ve výsledku dojde k rozostření hran a ke snížení šumu. Avšak je třeba podotknout, že rozostření hran nemusí být vždy

žádoucí jev, neboť dojde k odstranění malých detailů, které mohou být v jistém případě důležité. Výše zmíněná filtrace je ekvivalentem dolní propusti ve frekvenční oblasti.



Obrázek 2.1: 2D Gaussova distribuce se středem v bodě $(0, 0)$ a $\sigma = 1$

V praxi jsou vyhlazovací prostorové filtry reprezentovány tzv. *Gaussiánem*. Konkrétně jde o filtarční masku, jejíž hodnoty jsou dány aproximací Gaussova rozložení (viz obr. 2.1)
Průběh Gaussova rozložení je pak definován vztahem

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{2.6}$$

2.3 Prostorové filtry pro zostření obrazu

Filtrační metody pro zostření obrazu úzce souvisí s diskrétní podobou derivace. Jedná se prakticky o protějšek metod pro rozostření obrazu. Jsou tedy potlačovány místa v obrazu, kde se jas mění pomalu. Naopak jsou zvýrazňovány vyšší frekvence, čemuž odpovídá zvýraznění hran objektů. Ekvivalentem ve frekvenční oblasti jsou filtry typu horní propust.
Pro zostření se využívají zejména metody založené na první a druhé derivaci. Z pohledu vlastností je druhá derivace citlivější na prudké změny jasu. To má za následek kvalitnější detekci detailů v obrazu, ovšem za cenu vyšší citlivosti na šum v obraze. Tento fakt jsme schopni ovlivnit vhodným předzpracováním obrazu [7].

2.3.1 Metody založené na první derivaci

V případě první derivace dvourozměrné diskrétní funkce $f(x, y)$ mluvíme o gradientu fukce f v bodě (x, y).

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.7}$$

Jeho velikost spočteme následovně

$$\nabla f = \sqrt{G_x^2 + G_y^2} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{2.8}$$

Vyhledem k tomu, že je nezbytné pro stanovení gradientního obrazu provést operaci výpočtu velikosti vektoru (2.8) pro každý pixel, bývá tato operace zjednodušena na

$$\nabla f \approx |G_x| + |G_y| \,. \tag{2.9}$$

Pro návrh samotné filtrační masky je vhodné použít označení prvků filtrační masky a překrytých pixelů obrazu ze vztahu 2.5. Následně s použitím vztahu 2.9 a tzv. křížových diferencí definovaných L.G.Robertsem [7], dostáváme vztah [1]

$$\nabla f \approx |z_5 - z_9| + |z_6 - z_8| \,. \tag{2.10}$$

Dle uvedených koeficientů je zřejmé, že se nejedná o aproximace orientované v ose x a y, ale jedná se o aproximace diagonální. Ty mohou být chápány jako parciální derivace gradientu 2D obrazové funkce v souřadném systému s osami odpovídajícími daným diagonálním směrům. Můžeme tedy stanovit velikost výsledného vektoru jasového gradientu a dostáváme tak dvojici filtračních masek, tzv. *Robertsovy křížové operátory*, resp. *Robertsova konvoluční jádra*.

$$\underset{\text{směr 1}}{\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}} \quad \underset{\text{směr 2}}{\begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}} \tag{2.11}$$

Vzhledem k tomu, že tyto filtrační masky neobsahují střední koeficient (mají sudý počet řádků i sloupců) jsou obtížně implementovatelné. Tento problém však řeší Gradientní operátory dle Prewittové [1]

$$\nabla f \approx |(z_3 + z_6 + z_9) - (z_1 + z_4 + z_7)| + |(z_7 + z_8 + z_9) - (z_1 + z_2 + z_3)| \tag{2.12}$$

$$\underset{\text{směr x}}{\begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}} \quad \underset{\text{směr y}}{\begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}} \tag{2.13}$$

[1] Uvedený vztah platí pro filtrační matici o rozměru $n \times n$, kde $n = 3$.

Další variantou jsou tzv. Sobelovy operátory neboli Sobelova konvoluční jádra. Ty na rozdíl od předchozí metody ještě kladou důraz na střední pixel. Dojde tak k vyhlazení výsledné funkce

$$\begin{aligned}\nabla f \approx &|(z_3 + 2z_6 + z_9) - (z_1 + 2z_4 + z_7)| + \\ &+ |(z_7 + 2z_8 + z_9) - (z_1 + 2z_2 + z_3)|\end{aligned} \tag{2.14}$$

$$\begin{array}{cc} \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} & \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \\ \text{směr x} & \text{směr y} \end{array} \tag{2.15}$$

2.3.2 Metody založené na druhé derivaci

Nejjednodušším operátorem aproximujícím druhou derivaci obrazové funkce $f(x, y)$ je tzv. *Laplacián*. Ten je za pomoci parciálních derivací definovám jako

$$\nabla^2 f = \left[\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \right] \tag{2.16}$$

Pro aproximaci diskrétní podoby parciálních derivací druhého řádu pro osu x a y vyjdeme ze vztahu pro jednu proměnnou

$$\begin{aligned}\tfrac{\partial^2 f}{\partial x^2} &\approx [f(x+1) - f(x)] - [f(x) - f(x-1)] = \\ &= f(x-1) - 2f(x) + f(x+1)\end{aligned} \tag{2.17}$$

Z výše uvedeného vztahu pak dostáváme pro oba směry ve dvourozměrné obrazové funkci pro směr x

$$\frac{\partial^2 f}{\partial x^2} = z_4 - 2z_5 + z_6 \tag{2.18}$$

a pro směr y

$$\frac{\partial^2 f}{\partial y^2} = z_2 - 2z_5 + z_8 \tag{2.19}$$

Po dosazení dvou výše uvedených vztahů do 2.16 dostáváme

$$\nabla^2 f = z_2 + z_4 - 4z_5 + z_6 + z_8 \tag{2.20}$$

Vztah 2.20 můžeme implementovat pomocí filtrační masky následovně

$$\begin{array}{cc} \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} & \begin{bmatrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{bmatrix} \\ \text{Laplacián pro 4-okolí} & \text{směr Laplacián pro 8-okolí} \end{array} \tag{2.21}$$

Jak již bylo v úvodu řečeno, *Laplacián* potlačuje jas v oblastech, kde se jas mění pozvolna, naopak nespojitosti v obraze zvýrazňuje. Mluvíme tedy o derivativním operátoru. V konečném výsledku pak dostáváme obraz, jež má hrany a jiné nespojitosti světlé, ostatní oblasti s konstatním jasem jsou tmavé. Z výše uvedeného lze tedy vyvodit, že samotná aplikace *Laplaciánu* nevede k zostření hran, ale lze jí dosáhnout správným postupem. Tím je přičtení původního obrazu k výsledku filtrace pomocí *Laplaciánu*.

3 KNIHOVNY PRO ZPRACOVÁNÍ OBRAZU

Zpracováním obrazu rozumíme vědeckotechnickou disciplínu, která zpracovává digitální obrazová data různého původu: fotoaparát, kamera, ultrazvuk atd. Zpracování obrazu je podoborem digitálního zpracování signálu, jelikož se zabývá zpracováním 2D digitálních signálů. Obrazové knihovny poté slouží přímo pro práci s různými formáty obrazových dat (jpg, tiff, gif, png, bmp a další) a zprostředkovávají uživateli, resp. programátorovi operace nad těmito daty. Samotné knihovny se většinou liší funkcemi a platformou, pomocí které jsou vytvořeny.
Následující kapitola popisuje a srovnává dostupné knihovny pro zpracování obrazu. Jsou zmíněny jejich výhody, nevýhody, omezení a na konci kapitoly bude diskutována vybraná knihovna, jež bude dále v práci použita pro návrh konkrétních komponent genetického programování.

3.1 Obrazová knihovna JImage

ImageJ je věřejně dostupná knihovna pro zpracování obrazu inspirovaná NIH (National Institutes of Health) Image knihovnou pro Macintosh. Funguje jak v online formě tak jako konzolvá alikace, požadavkem je Java Virtual Machine v1.1 či vyšší. Ke stažení jsou dostupné distribuce pro Windows, Mac OS, Mac OS X či Linux.
Samotná aplikace umožňuje zobrazovat, editovat, zpracovávat, ukládat a tisknout 8-bitové, 16-bitové a 32-bitové obrázky. Podporovány jsou formáty TIFF, GIF, JPEG, BMP, DICOM, FITS, RAW. Je možné využít dávek pro hromadné zpracování souborů. Celá aplikace je programována jako vícevláknová, tzn. časově náročnější operace, jako např. otevírání obrázků do okna, se vykonávají paralelně s ostatními méně náročnějšími operacemi.
Uživatel může výužít nejrůznějších operací nad obrazovými daty, např. měřit vzdálenosti, úhly, zobrazovat histogramy, manipulovat s kontrastem, zvýrazňovat hrany a monoho dalšího. Obrazy samotné je možno zvětšovat až do poměru 32:1 naopak zmenšovat až do poměru 1:32. V tomto rozmezí je možné na obrazová data aplikovat libovolné, výše zmíněné, operace. Lze pracovat s více obrazy současně, jediným limitujícím faktorem v tomto případě je dostupná paměť počítače.
Knihovna ImageJ byla vyvinuta jako open source projekt a veškeré zdrojové kódy jsou tak přistupné veřejnosti. Její hlavní síla tkví v rozšířitelnosti pomocí pluginů. Není-li tedy požadovaná funkce v jádru systému ImageJ je vysoká pravděpodobnost, že je realizována jako plugin. A pokud není k dispozici ani plugin je možné si ho vytvořit pomocí programovacího jazyka JAVA [11].

3.2 Obrazová knihovna ImLab

ImLab je opět open source projekt určený pro vědecké zpracování obrazů. Je dostupný pro platformy Windows, Linux a nejrůznější Unixové odnože. Stejně jako ImageJ umožňuje mnoho operací nad obrazovými daty, podporuje dávky a nejrůznější obrazové formáty. Zásadní rozdíl je ve struktuře a jazyku v kterém byl vytvořen. ImLab je implementován v jazyce C++ a C. Většina zdrojového kódu však není psána klasickou objektovou formou, kde jsou využívány třídy a z nich tvořeny instance. Autor využívá objektových vlastností C++ pouze pro implementaci některých obslužných procedur v jádru systému. Výsledný kód se pak jeví spíše jakoby byl psaný v jazyce C s drobnými rozšiřujícími funkcemi. Tato struktura má hlavní výhodu v rychlosti zpracování dat a celý projekt by tak měl být co nejsrozumitelnější pro případného programátora, jež s ním hodlá pracovat.
Struktura celého programu se skládá ze tří nejzásadnějších částí, jsou to:

- IUP (Portable User interface) - jedná se o soubor nástrojů pro tvorbů grafického uživatelského rozhraní. Je realizován pomocí API knihovny, jež poskytuje více jak 100 funkcí pro vytváření a manipulaci s dialogovými okny.
- CD (Canvas Draw) je grafická knihovna nezávislá na platformě využívající dostupné grafické knihovny: Microsoft Windows (GDI a GDI+) a X-Windows (XLIB). Knihovna podporuje jak vektorové, tak rastrové grafické obrazové aplikace.
- IM (Image Representation, Storage, Capture and Processing) je sada nástrojů pro digitální zpracování obrazů. IM se stará o následující 4 obrazové operace: reprezentace obrazu, skladování, zpracování a zachytávání dat. Hlavním cílem knihovny je poskytnout jednoduché API a abstrakci obrazů pro vědecké aplikace. Podporovány jsou následující formáty: TIFF, BMP, PNG, JPEG, GIF a AVI [13].

Knihovna ImLab neposkytuje obecně používané mechanismy pro rozšíření pomocí plug-inů, nicméně je možné si tvořit vlastní funkce, které se následně dají implementovat do systému, jsou přenositelné a prakticky se chovají jako plug-in.

3.3 Obrazová knihovna UTHSCSA ImageTool

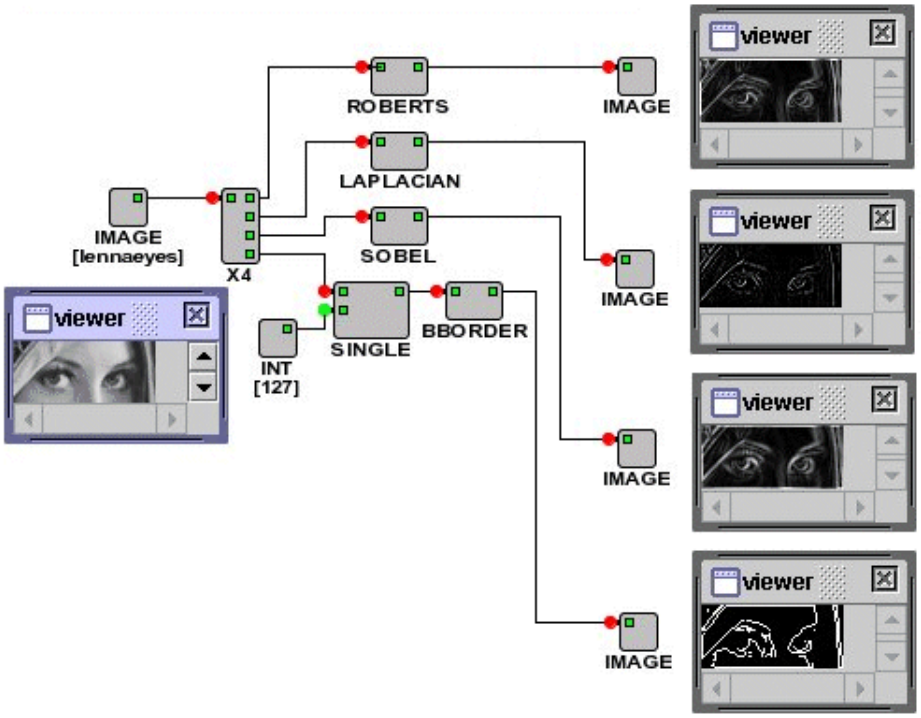
UTHSCSA ImageTool je volně šířitelný program pro zpracování a analýzu obrazu. Jeho zásadní nevýhodou je použitelnost pouze na platformě Windows (Windows 9x, Windows ME nebo Windows NT). ImageTool může otevírat, zobrazovat, upravovat, analyzovat, zpracovávat, komprimovat, ukládat a tisknout jak v odstínech šedi, tak

i barevná obrazová data. Podporuje více než 22 společných formátů souborů včetně nejpoužívanějších BMP, PCX, TIF, GIF a JPEG. Dále je možné analyzovat obraz pomocí měření vzdáleností, úhlů, plochy či aplikovat základní operace jako je změna kontrastu, detekce hran, vyhlazení atd. Je také možné využít konvolučních filtrů, kde konvoluční matici může definovat sám uživatel [14]. Všechny zmíněné oprace lze pomocí skriptů aplikovat jako dávku.
ImageTool byl navržen s otevřenou architekturou, která poskytuje rozšiřitelnost prostřednictvím různých plug-inů. Podpora pro snímání obrazu se používá buď v programu Adobe Photoshop opět jako plug-in nebo Twain scanner. Vlastní skripty pro analýzu, či zpracování obrazu lze vyvíjet pomocí tzv. Software Development Kit (SDK), jež je k dispozici (včetně zdrojového kódu). Tento přístup umožňuje řešit téměř jakoukoliv implementaci dat z externích zařízení, včetně její analýzy.
Přes velký počet funkcí a možnosti rozšířit pomocí plug-inů, není tato knihovna příliž používána díky své omezenosti prakticky na jednu platformu. Další zásadní nevýhodou je špatná podpora a fakt, že jeden z posledních updatů proběhl v květnu roku 2002, kdy byla vydána verze 3.0.

3.4 Obrazová knihovna NeatVision

NeatVision je vizuální programovací prostředí pro zpracování obrazových dat, založené na programovacím jazyku JAVA. Vývojové prostředí poskytuje intuitivní rozhraní pro řízení aplikací a procedur nad obrazovými daty. To je dosaženo pomocí blokových diagramů a drag and drop technologie. Každá operace je prakticky reprezentována jedním grafickým blokem se vstupy a výstupy. Tyto bloky je možné řetězit a navzájem mezi sebou propojovat, editovat, či mazat. Princip této práce je patrný z obr. 3.1. Na vstup je předán obrázek, pomocí čtyř cest je obrázek přefiltrován patřičným filtrem a následně zobrazen na výstup.
NeatVision je k dispozici ve dvou verzích. První uživatelská verze umožňuje vytvářet jednotlivé procesy pro obrazové zpracování pomocí svého grafického rozhraní a implementovaných funkcí (NeatVision v současné době obsahuje více než 290 funkcí použitelných pro zpracování a analýzu obrazu, od pixelových operací po manipulaci s barevnými prostory). Pro vývojáře je dostupná verze, která umožňuje uživatelům integrovat své vlastní funkce, tj. na modernizaci zavedením nových modulů pro zpracování obrazu.
Ve výsledku je NeatVision velice mocný nástroj s poměrně inteligentním grafickým prostředím pro digitální zpracování obrazu, nicméně je poměrně komplikovaný na instalaci. Další nevýhodou je nutnost specifické verze JRE (Java Runtime Environment). A stejně jako u knihovny ImageTool již dlouho neproběhl update, poslední

verze 2.1 byla vadána v říjnu roku 2003 [12]. V dnešní době je tedy další vyvoj prakticky zastaven.



Obrázek 3.1: Princip práce s knihovnou NeatVision

3.5 Výběr obrazové knihovny pro další práci

Pro další pokračování v práci byla vybrána obrazová knihovna JImage. První její zásadní výhodou je, že je stále aktualizována a celý projekt je uživatelsky aktivní. Dalším aspektem, proč použít zmíněnou knihovnu, je návaznost na projekt evolučního systému vyvýjeného na VUT FEKT Brno v jazyce JAVA. Z toho vyplívají výhody při implementaci jednotlivých komponent pro evoluční systém a celková kompatibilita projektu.

4 PRÁCE S KNIHOVNOU JIMAGE

V následující kapitole bude popsána práce s knihovnou JImage z uživatelského a programátorského hlediska. Její instalace, práce se základními komponentami a bude demonstrován způsob, jakým lze rozšířit funkce přidáním pluginu. Z programátorského pohlediska je důležitý průzkum vnitřní struktury, aby bylo možné efektivně pracovat s dostupnými třídami, objekty, apod. Bez těchto znalostí by nebylo možné vyvíjet vlastní komponenty, jež obsahem poslední kapitoly této práce. Pro bezproblémový chod knihovny potřebujeme mít k dispozici třídu JImage.class, nezbytné konfigurační soubory, JRE (Java Runtime Environment) a pro kompilování vlastních plug-inů Java compiler s potřebnými knihovnami (Java 2 SDK Standard Edition - J2SE) [15][16].

4.1 Instalace ImageJ

Aktuální verze ImageJ je k dispozici na stránce `http://rsb.info.nih.gov/ij/download/`. V případě, že uživatel nepracuje s programovacím prostředím JAVA a nemá nainstalované JRE a JAVA Compiler bude muset pro instalaci postupovat podle návodu na oficiálních webových stránkách `http://rsb.info.nih.gov/ij/docs/install/`. V případě máme-li nainstalované uvedené komponenty stačí stáhnout Imagej.class a její konfigurační soubory, jež jsou dostupné v zip archívu. Pro spuštění pak již stačí pouze přidat soubor ij.jar a nastavit správně cestu pro spuštění třídy ij.ImageJ.

4.2 Struktura adresáře ImageJ

Po správné instalaci dostaneme strukturu uvedenou níže, její základní znalost je nezbytná pro další rozšiřování, např. pomocí plug-inů.

- `/jre` - zde jsou umístěny veškeré soubory nezbytné pro vykonávání programů psaných v jazyce JAVA, tzv. Java Virtual Machine (JVM).
- `/macros` - adresář obsahuje makra pro ImageJ, jedná se o krátké programy psané v JImage macro jazyku.
- `/plugins` - adresář pro nejrůznější pluginy vytvořené uživatelem. Jedná se o soubory *.class, ty můžeme tvořit samy, popřípadě stáhnou z oficiálních stránek[1] JImage.

[1] `http://rsb.info.nih.gov/ij/plugins/index.html`

- `/ij.jar` - JAVA archív obsahující základní funkcionalitu ImageJ, prakticky se jedná o jádro celého projektu v binární podobě. Při vydání nové verze ImageJ stačí pouze přepsat jádro ij.jar a tím docílíme upgradu na nejnovější verzi. Ostatní funkcionalita dodaná po instalaci (např. pluginy, makra) samozřejmě zůstane zachována.
- `/IJ_Prefs.txt` - v uvedeném souboru jsou uloženy nejrůznější nastavení a informace o uživatelském rozhraní ImageJ - poslední otevřené soubory, velikosti jejich oken, výchozí barva pozadí okna, apod.
- `/ImageJ.cfg` - specifikuje cestu pro spuštění Java runtime procesů. Pro Windows bývá standartně použito následující nastavení:

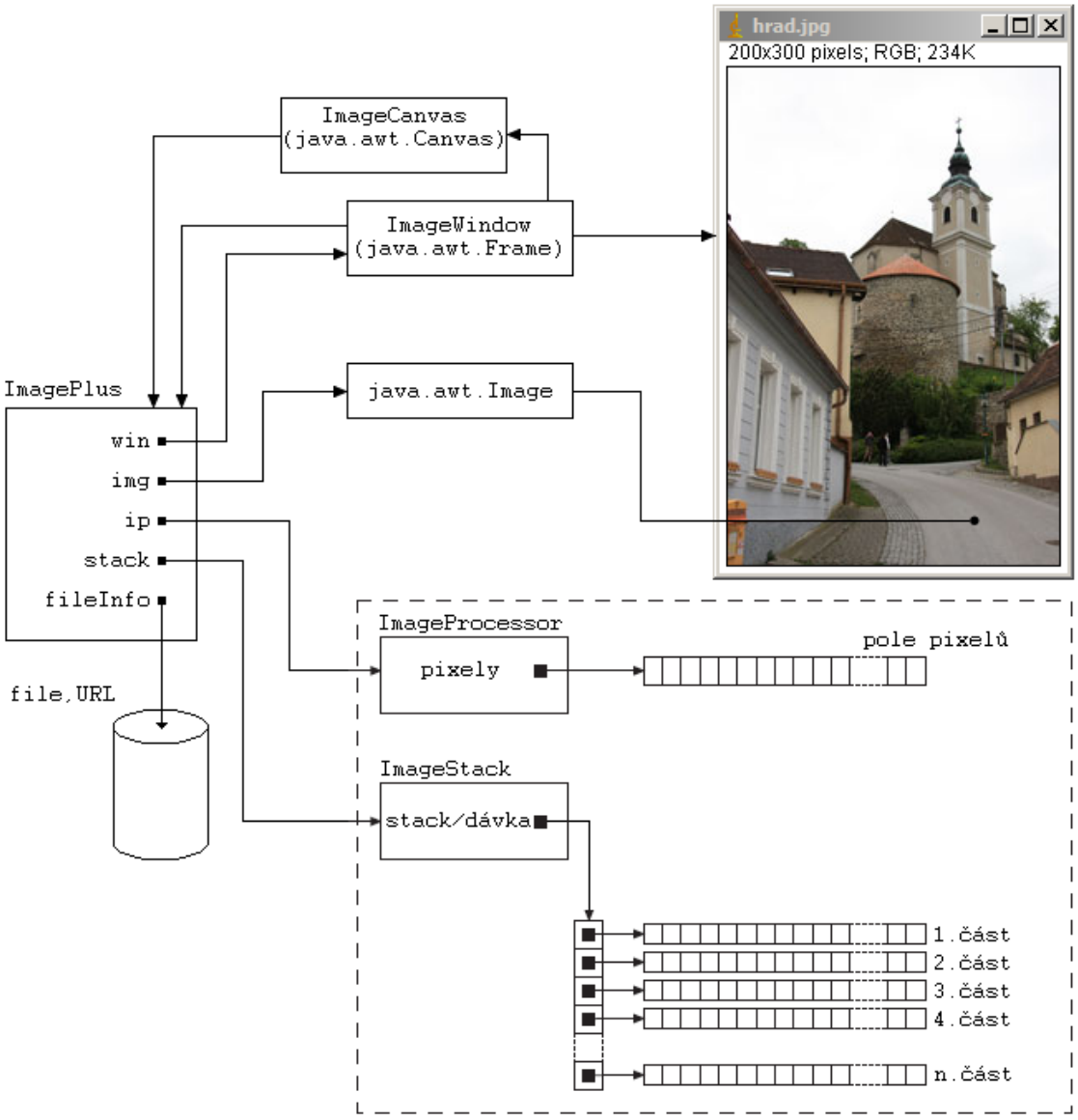
  ```
  .
  jre\bin\javaw.exe
  -Xmx340m -cp ij.jar ij.ImageJ
  ```

  kde `-Xmx340m` je parametr pro nastavení paměti alokované pro Java procesy. V tomto případě je alokována paměť 340MB. To však může být pro některé případy relativně malá hodnota, je tedy doporučeno paměť zvýšit na 640MB. Pokud by i to bylo málo pro některé procesy, lze alokaci paměti zvýšit až na 1,7GB paměti pro 32-bitové systémy.
- `/ImageJ.exe` - jedná se o spouštěcí program pro Windows.

4.3 Vnitřní struktura ImageJ

Pro jakoukoliv programátoskou činnost s využítím instancí, objektů a metod odvozených z knihovny JImage je nezbytná znalost její vnitřní struktury. Ta nám následně umožní přístup k obrazovým datům, realizuje vykreslení, otevírání, resp. ukládání souborů do požadovaných formátů apod. Zjednodušené schéma pro popis práce s obrazovými daty je znázorněn na obr. 4.1. Struktura se praticky skládá ze tří hlavních částí:

- Instance `ImageProcessor` - jejím úkolem je udržovat a poskytovat přístup k samotným pixelům obrazu.
- Instance `java.awt.Image` se stará o vykreslování obrazu do okna.
- Instance `ImagePlus` se stará o udržování veškerých metadat (popisky, vlastnosti, apod.) výše zmíněných tříd `ImageProcessor` a `java.awt.Image`.



Obrázek 4.1: Vnitřní struktura ImageJ

4.3.1 Instance ImageProcessor

Jak už bylo uvedeno výše instance `ImageProcessor` uchovává aktuální obrazová data (pixely). Ty jsou uloženy ve formě jednorozměrného pole. V případě instance `ImageStack` jsou data uložena taktéž v polích, ovšem tentokráte chronologicky jdoucími po sobě, neboť se jedná o více obrázků v řadě, tzv. dávku. Jde prakticky o frontu FIFO. Samotná instance `ImageStack` je tedy totožná s `ImageProcessor`, liší se jen uchováním více snímků.

Z hierarchického hlediska představuje `ImageProcessor` nadřazenou abstraktní třídu, která poskytuje svým odvozeným třídám patřičné metody pro práci s požadovanými formáty. Jedná se o:

- `ByteProcessor` (class) - pracuje s 8-bitovými (byte) indexovanými formáty ve stupních šedi. Hodnoty pixelů v této třídě tak mohou nabývat hodnot v rozmezí 0 až 255.
- `ShortProcessor` (class) - pracuje s 16-bitovými obrazy ve stupni šedi.
- `FloatProcessor` (class) - slouží pro práci s 32-bitovými obrazovými daty. Jedná se o datový typ s plovoucí řádovou čárkou.
- `ColorProcessor` (class) - už z názvu se dá vyvodit, že jde o třídu určenou pro práci s barevnými obrázky. Obsahuje 3 × 8 bitů pro jednotlivé složky RGB plus dalších 8 bitů pro alfa kanál.

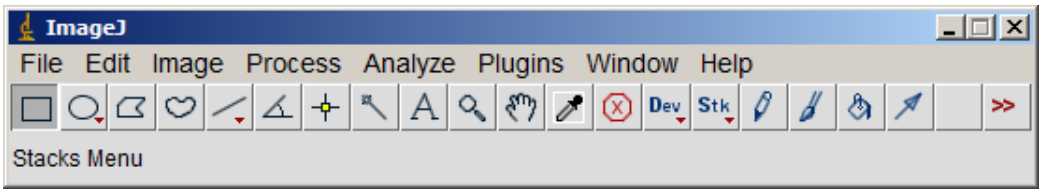
Můžeme tedy konstatovat, že instance `ImageProcessor` nám poskytuje nástroj, jak zpracovávat aktuální obrazová data. Nedokáže je však graficky interpretovat. O to se stará instance `ImagePlus`.

4.3.2 Instance ImagePlus

Jedná se o rozšířenou variantu standardní třídy `java.awt.Image` určenou pro reprezentaci obrazu. Z kontextu tedy vyplívá, že vytvořením objektu z třídy `ImagePlus` obdržíme metody starající se o vykreslování dat na obrazovku monitoru.

4.4 Uživatelské úpravy obrazů

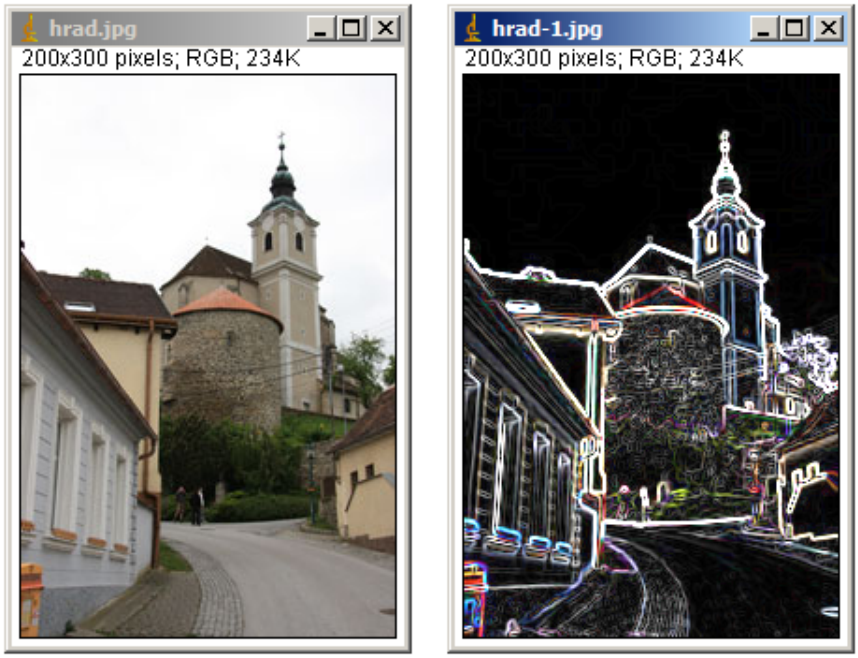
Samotné uživatelské úpravy obrazů jsou pak relativně triviální. Děje se přes uživatelské rozhraní, zobrazené na obr. 4.2.



Obrázek 4.2: Uživatelské rozhraní knihovny ImageJ

Pro demostraci funkčnosti otevřeme obrázek ve formátu *.jpg a pokusíme se u něj zvýraznit hrany.
Výše uvedené realizujeme přes kontextové menu *File* ⇒ *Open* a nastavíme cestu na HDD k požadovanému obrázku. Nyní máme k dispozici obrázek v nově otevřeném okně a můžeme na něj aplikovat požadované operace. V našem případě zvýrazníme

hrany. Opět přes kontextové menu *Process* ⇒ *Find Edges*. Výsledek je možné si prohlédnout na obr 4.3



Obrázek 4.3: Nalezení hran pomocí JImage

4.5 Implementace plug-inů v JImage

Implementace plug-inů je taktéž poměrně triviální záležitostí. Pro demostraci stáhneme a znázorníme např. 2D Gaussův filtr. Zdrojový kód stáhneme z oficiálních webových stránek [2] JImage. Zde máme dále k dispozici informace o autorovi daného plug-inu, jeho verzích, popis k čemu slouží a v neposlední řadě jsou zde doplněny i pokyny pro instalaci. Ta většinou obnáší překopírovaní zdrojového kódu do složky s jednotlivými plug-iny, vždy je však nezbytné tyto informace pročítat, neboť požadovaný plug-in může být přímo závislý i na jiném. Nakonec většinou stačí ImageJ restartovat a následně máme k dispozici požadovaný plug-in přes kontextové menu.

Pro aplikaci zmíněného 2D Gaussova filtru tedy stáhneme soubor `Gaussian_Filter.class` a nahrajeme do adresáře `/plugins`. Restartujeme aplikaci a nyní pomocí kontextového menu *Plugins* ⇒ *Filters* ⇒ *Gaussian Filter* dostaneme průběh Gaussova filtru pro standartní rozložení, viz. obr 4.4.

[2] `http://rsb.info.nih.gov/ij/plugins/index.html`

Obrázek 4.4: Implementace plug-inu Gaussova filtru v ImageJ

5 VYTVOŘENÍ KOMPONENT PRO GENETICKÉ PROGRAMOVÁNÍ

V následující kapitole bude představeno šest komponent pro genetické programování. Budou představeny metody filtrace, které nabízí knihovna JImage, dále pak návrh filtrů pomocí konvoluce včetně diskuze rozdílů mezi zmíněnými přístupy. Pro zvýšení efektivity obrazového zpracování byly vytvořeny masky fungující analogicky jako masky vrstev v programech Photoshop, či GIMP. U této problematiky budou probrány různé technologie pro vytvoření masek, dikutovány jejich výhody, nevýhody a omezení. Pro lepší přehlednost a efektivitu práce byla vytvořena komponenta pro načítání dat, v našem případě obrazů a masek pro následné zpracování.

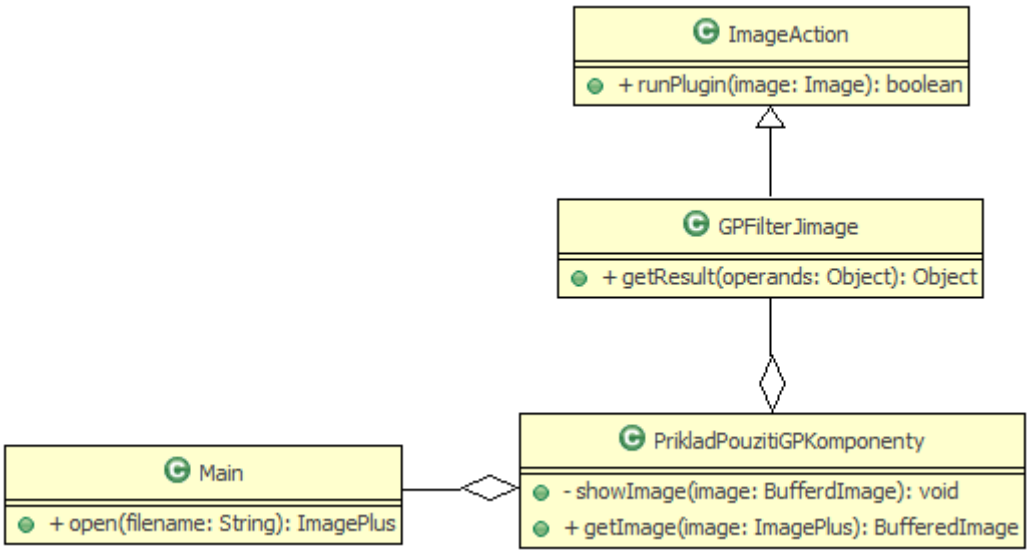
5.1 Filtrace s využitím metod JImage

První a nejjednodušší možností realizace obrazových filtrů je využití metod dostupných přímo v JImage. Máme k dispozici poměrně široké spektrum metod pro zpracování obrazu, konkrétně od detekce hran přes mediánový filtr až po možnost přidat do obrazu náhodný šum. Můžeme také matematicky manipulovat s jednotlivými pixely obrazu. Např. přičítat, odčítat, násobit, či umocňovat hodnoty pixelů. Obraz můžeme také invertovat apod. Výčet těch nejpoužívaněších metod jest uveden níže [15].

- `void convolve3x3(int[] kernel)` - provede konvoluci s čtvercovým jádrem o rozměru 3.
- `void sharpen()` - provede zostření obrazu pomocí konvoluce s matící o rozměru 3×3.
- `void smooth()` - nahradí aktuální pixel průměrnou hodnotou vypočtenou z okolních pixelů v teritoriu 3×3.
- `void noise(double range)` - přidá do snímku náhodný šum, jeho velikost je dána vstupním parametrem `range`.
- `void findEdges()` - zvýraznění hran pomocí již předpřipraveného Sobelova operátoru.
- `void medianFilter()` - medián filtr 3×3.
- `void gamma(double value)` - provede gamma korekci, její velikost je definována vstupním parametrem `value`.

- `void invert()` - provede inverzi všech pixelů v obrazu.
- `void add(int value)` - ke všem pixelům přičte definovanou hodnotu.
- `void multiply(int value)` - každý pixel vynásobí s danou hodnotou.
- `void sqr()` - každý pixel je umocněn na druhou.
- `void sqrt()` - na každý pixel je aplikována druhá odmocnina.

Výše zmíněné metody jsou využitelné v první komponentě pro genetické programování. Ta je reprezentována třídou `GPFilterJimage`. Na jejím vstupu je předán obraz jako datový typ `ImagePlus` z něhož pomocí instance `ImageProcessor` získáme přístup k jednotlivým pixelům. Nyní nám již nic nebrání v tom použít jednu z výše uvedených metod. Ty lze jednotlivě po sobě kombinovat, či řetězit. Výstupem je následně obraz datového typu `image`, jež může být zobrazen na výstup. To se následně děje ve třídě `PrikladPouzitiGPKomponenty`. Celá situace je lépe zřetelná z UML digramu (obr. 5.1), jež celou aplikaci dokumentuje.



Obrázek 5.1: UML diagram reprezentující GP komponentu pracující s filtry dostupný mi z knihovny JImage

Z uvedeného plyne, že výše zmíněné metody jsou poměrně jednoduché na implementaci, avšak díky své neuniverzálnosti jsou pro genetické programování téměř nepoužitelné. Drtivá většina metod založených na konvoluci je zpracovávána pevně přednastaveným jádrem 3×3, čímž jsou prakticky jednoúčelné.

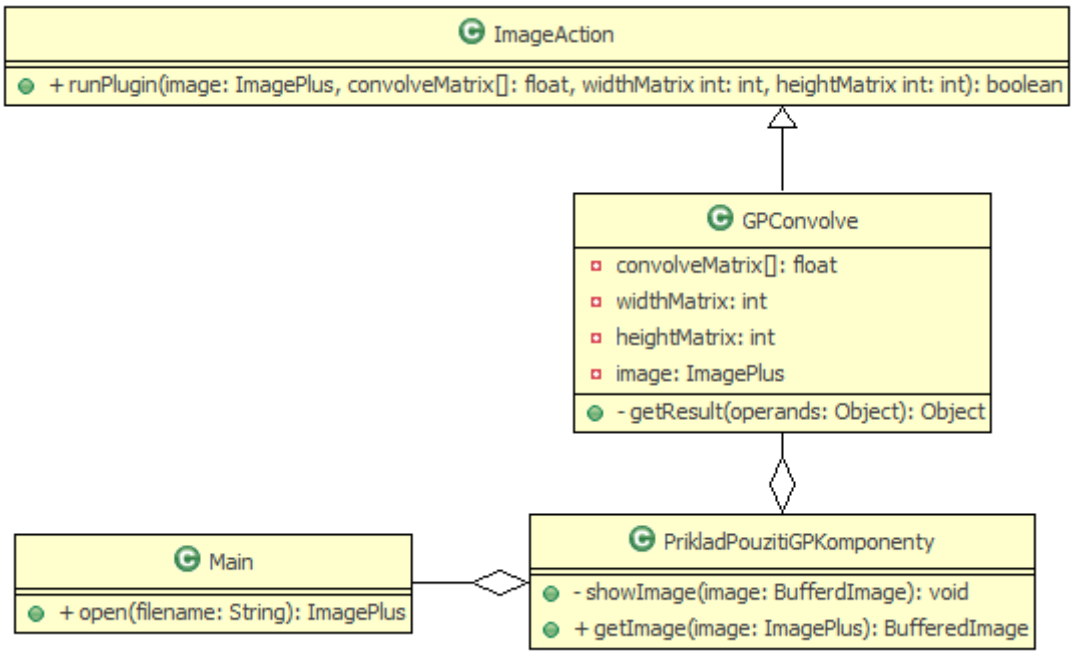
5.2 Komponenta pro vytvoření konvolučního filtru

Z předchozí kapitoly je patrné, že pro návrh filtrů pomocí evolučních algoritmů bude třeba zvolit jiný přístup. A to takový, jež bude natolik univerzální, aby samotný systém mohl navrhovat jednotlivé filtry, ty následně testovat a vyhodnocovat jejich použitelnost pro daný zdroj. Je tedy nezbytné přistoupit k danému problému zcela univerzálně.
Pro co nejuniverzálnější postup bylo třeba zajistit technologii, pomocí které systém vytvoří libovolný obrazový filtr. Pro tento požadavek byla zvolena metoda 2D konvoluce. Jako vstupní parametry je třeba zadat zdrojový obraz a definovat jádro konvoluční matice. To provedeme pomocí tří parametrů: šířky matice, výšky matice a naplnění matice konkrétními hodnotami. Obraz pro zpracování lze principielně načíst jak z místního disku, tak z URL. Jeho formát musí být samozřejmě kompatibilní s podporovanými formáty v JImage. Výstupem je pak upravený obraz paptřičným filtrem.

Obrázek 5.2: Principielní schéma komponenty pro návrh konvolučního filtru.

Nyní se zaměřme na funkčnost celé komponenty z programátorského hlediska. Základním stavebním kamenem celé funkcionality je třída `GPConvolve`. Ta obsahuje metodu `getResult` se vstupními parametry dle schématu 5.2, resp. obraz, který se bude zpracovávat, šířka a výška jádra konvoluční matice a její hodnoty. Pomocí instance `ImageProcessor` získáme přístup k jednotlivým hodnotám pixelů, ty jsou teď přístupné v poli hodnot a můžeme tak přímo provést konvoluci s definovaným jádrem. Obraz je následně updatován a jsou předána na výstup upravená obrazová data.
Abychom mohli celou aplikaci otestovat, byla vytvořena třída `PrikladpouzitiGPKomponenty`, kde naplněním vstupních dat a vytvořením instance z třídy `GPConvolve` můžeme realizovat prakticky libovolnou 2D konvoluci. O zobrazení výsledků se starají dvě metody: `getImage` a `showImage`. První zmíněná zajišťuje konverzi obrazových dat do datového typu `BufferedImage`. Ten je

následně vstupním parametrem druhé metody, která se stará o samotné vykreslení do okna. Avšak dříve než začneme cokoli vykreslovat, musíme načíst požadovaná data. To se děje vytvořením instance třídy `Main`, která obsahuje metodu `open` pro načtení zdrojového obrázku. Celý proces je dokumentovám UML diagramem obr.: 5.3.



Obrázek 5.3: UML diagram komponenty pro návrh konvolučního filtru

5.2.1 Návrh a implemetntace konkrétních obrazových filtrů.

Níže budou demostrovány některé ze získaných výsledků při návrhu konkrétních obrazových filtrů. Jak již bylo uvedeno filtrace obrazu znamená 2D konvoluci jádra s každým z kanálu obrazu *f*. Filtr postupně zpracovává všechny pixely obrázku. Pro každý z nich vynásobí hodnotu aktuálního pixelu a jeho osmi sousedních pixelů odpovídajícími hodnotami jádra (platí pro jádro 3×3). Výsledné hodnoty sečte a výsledek je pak hodnotou přiřazenou aktuálnímu pixelu. Výše uvedený postup lze matematicky vyjádřit rovnicí 5.1 [17].

$$g[x,y] = \sum_{row=0}^{M-1} \sum_{row=0}^{N-1} h[row,col] \cdot f[x - xpos + row, y - ypos + col] \tag{5.1}$$

Zostření - horní propust

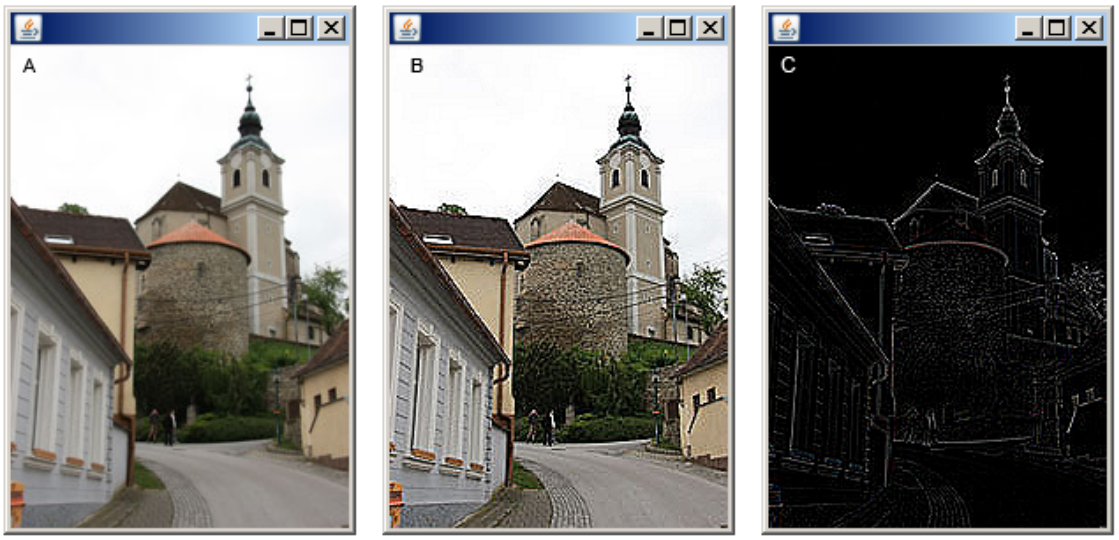
Zostření neboli horní propust můžeme realizovat např. pomocí Laplaceova operátoru, při jeho aplikaci dojde k zvýraznění horních kmitočů v obrazu. Jádro (Kernel) bude

mít tvar dle matice *h1*. Další možností je využití *Laplaciánu*[7]. V tomto případě ovšem nedostaneme zostřený obraz, nýbrž dojde k potlačení jasu v oblastech, kde se jeho hodnoty mění pozvolna (tmavé plochy) zatímco v oblastech rychlých změn jasu dojde k jeho zvýraznění (světlé plochy). Zostřený obraz dostame po sečtení původního obrazu s výlsledkem po aplikaci *Laplaciánu*.

$$h1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & -1 & 5 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad h2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & -4 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.2}$$

Laplaceův operátor Laplacián

Výsledek celé operace je patrný z obr. 5.4



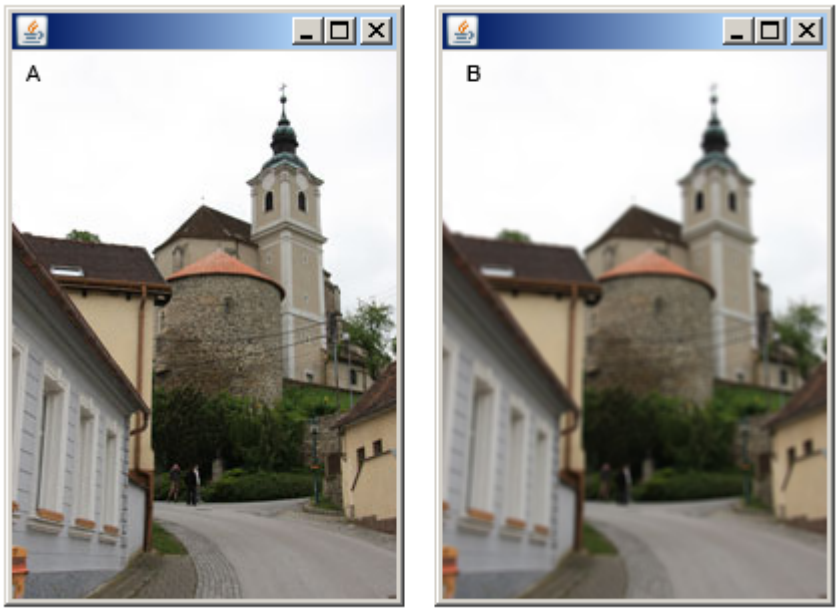
Obrázek 5.4: A - originální obrázek, B - po aplikaci horní propusti, C - Výsledek po aplikaci Laplaciánu

Rozmazání - dolní propust

Rozostřením dojde k potlačení vyšších kmitočtů, což má za následek rozmazání hran, resp. detailů v obraze. Filtr se využívá např. pro odstranění šumu obsaženého ve snímcích. Ovšem jak již bylo uvedeno dojde také k odstranění detailů, což může být v některých případech kontraproduktivní. použité jádro má tvar dle matice 5.3.

$$h = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.3}$$

Výsledek celé operace je patrný z obr. 5.5



Obrázek 5.5: A - originální obrázek, B - po aplikaci dolní propusti

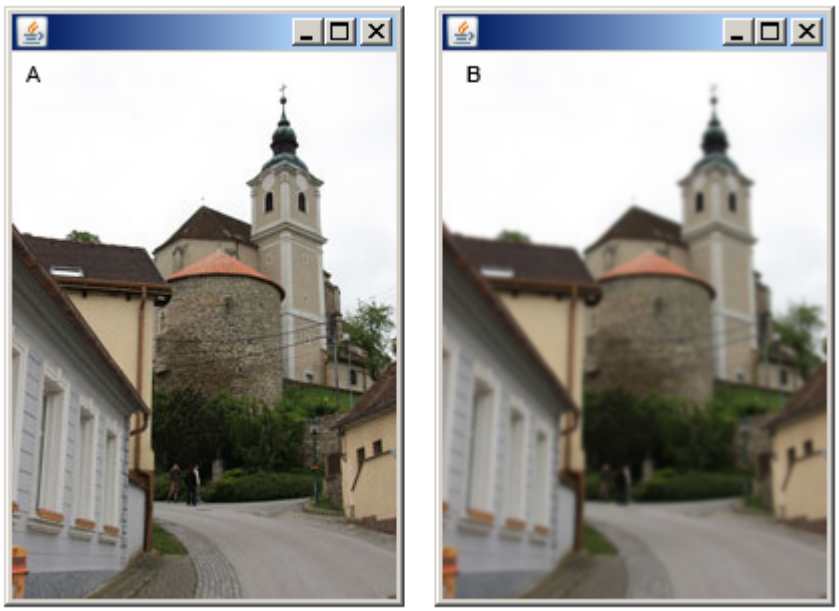
Pozn. v případě, že budou v matici 5.3 jedničky obsaženy na všech pozicích, bude obraz rozmazán ještě znatelněji.

Gaussův filtr

Gaussův filtr slouží pro vyhlazení obrazu, resp. pro jeho rozostření. To je dáno zejména velikostí směrodatné odchylky σ. Používá se opět pro odstranění šumu v obrázcích. použité jádro má tvar dle matice 5.4.

$$h = \begin{bmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 47 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{bmatrix} \tag{5.4}$$

Výsledek celé operace je patrný z obr. 5.6



Obrázek 5.6: A - originální obrázek, B - po aplikaci Gaussova rozostření

Detekce hran pomocí Sobelova oparátoru

Detekci hran v obraze používáme pro nalezení obrysů či hran. Jedná se obvykle o velkou změnu jasu, odstínu nebo stupně šedi na relativně malém intervalu. Abychom byli co nejpřesnější, pomocí Sobelova operátoru, můžeme detekovat hrany, jak ve vodorovném směru, tak v horizintálním směru. Jádro pro detekci hran v ose x je popsáno maticí 5.5a, pro detekci hran v ose y je popsáno maticí 5.5b.

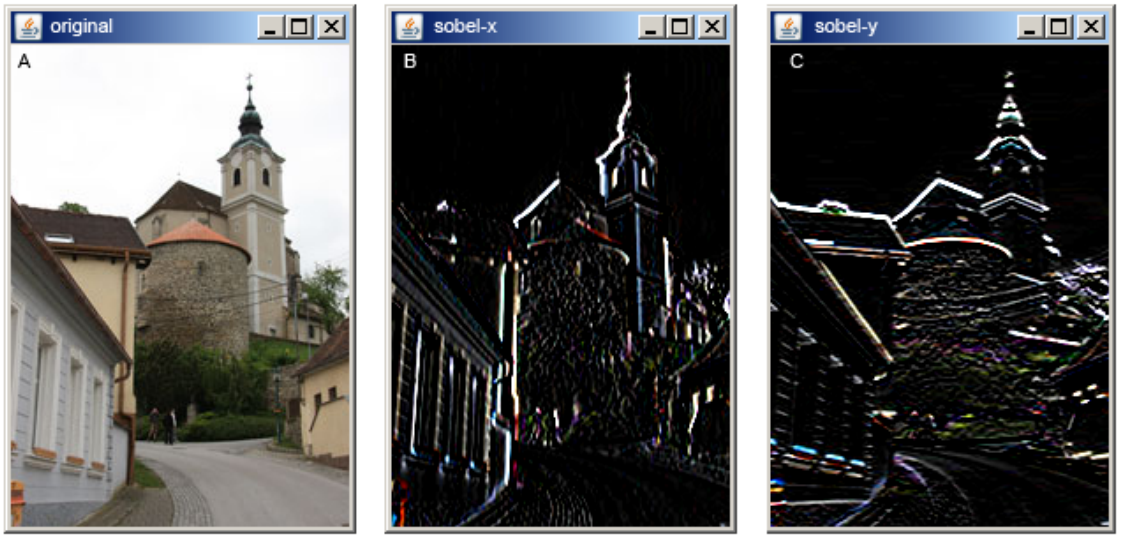
$$h1 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 2 & 0 & -2 & 0 \\ 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad h2 = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.5}$$

a - detekce hran v ose x b - detekce hran v ose y

Výsledek celé operace je patrný z obr. 5.7

Další vývoj práce - implementace masky obrazu

Nyní máme k dispozici komponentu pro genetické programování, jež je schopna navrhnout konvoluční obrazový filtr. Systém tedy může zkoušet na daný zdroj různé typy filtrů, popř. jejich kombinace a vyhodnocovat jejich relevanci. Mnohdy se však stává, že u daného obrazu chceme pracovat pouze s jeho částí, jeho zbytek pro nás tvoří relativně nezajímavou část. Ta může mnohdy zabírat více jak 50% dat.



Obrázek 5.7: A - originální obrázek, B - po aplikaci Sobelova operátoru v ose x, C - po aplikaci Sobelova operátoru v ose y

Představme si třeba Roentgenův snímek kosti. Budeme-li chtít zvýraznit například zlomeninu s jejím okolím, zjístíme, že podstatná část snímku je pro nás nezajímavá. Evoluční systém ovšem dostane úkol nálezt ideální filtr, popřípadě kombinaci filtrů, jež nám danou zlomeninu zvýrazní. Bude tedy navrhovat různé obrazové filtry a podle nastavení systému testovat relevantnost obdržených dat. To se bude ovšem dít přes celý obraz, resp. matici hodnot, což povede k poměrně neefiktivnímu využití výpočetní síly. Zvážíme-li nadbytečnost dat 50% je zřejmé, že kdybychom dokázali pracovat pouze s částí která nás zajímá ušetříme polovinu času. Stačilo by nám tedy teoreticky na vstupu kromě vstupních dat také předat masku pro daný obraz, jež by nám specifikovala náš obsah zájmu a se zbytkem dat by se již dále nepracovalo. Zmíněným postupem se proto zabývá zbylá část práce, jsou uvedeny další čtyři komponenty pro aplikaci masky. Jsou zmíněny jejich výhody, nevýhody a omezení.

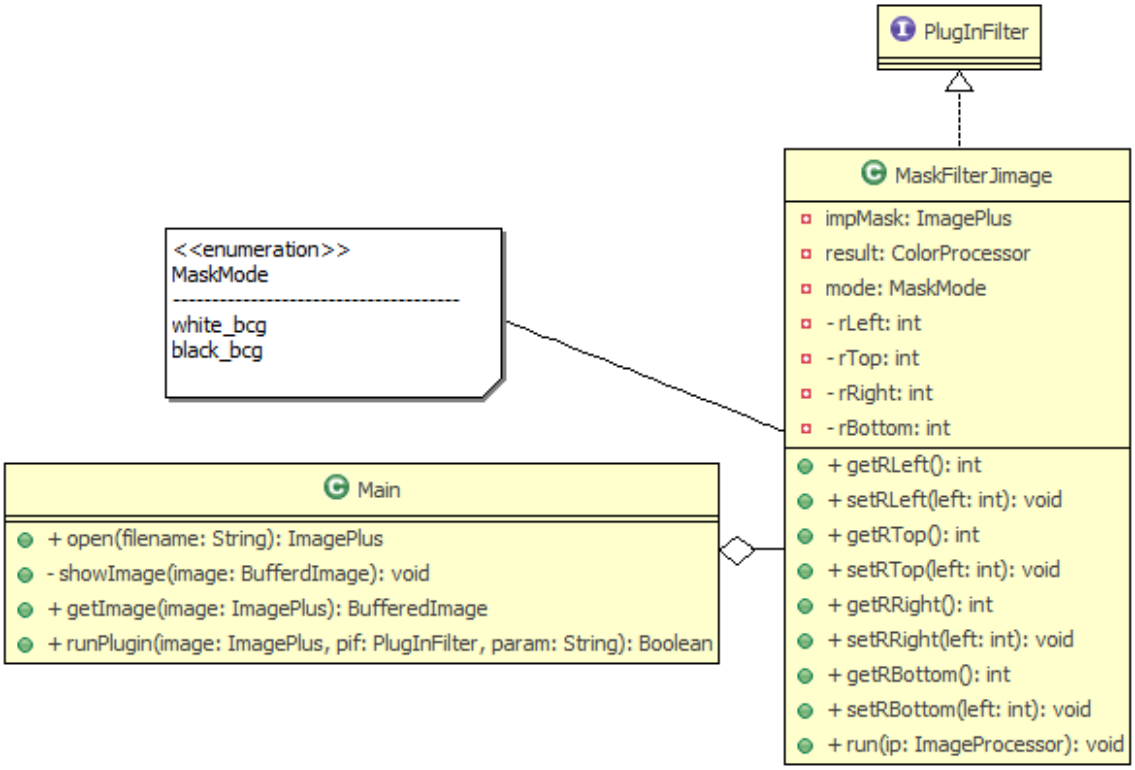
5.3 Implementace masky dostupné v knihovně JImage

První možností jak implementovat masku vrstvy je využít přímo metod knihovny JImage. Postup je poměrně jednoduchý, ale jak to tak bývá v těchto případech skýtá určitá omezení. Výhodou je, že můžeme pracovat s barevnými obrázky, tzn. se třemi kanály RGB. Prvním omezením je možnost pracovat pouze s omezenými tvary. Knihovna JImage nabízí klasické základní geometrické tělesa: čtverec, obdélník, polygon, kružnice, elipsa. Složitější tělesa nelze jednoduše dodefinovat. Další nevýhodou

je absence alfa kanálu, tzn. nedosáhneme zamaskování např. 50%, což může být výhodné v případech, kdy chceme danou informaci pouze potlačit, nikoliv odstranit. Je třeba se také zamyslet nad tím, jak vlastně maska nad obrazem bude fungovat, jak bude definována a co pro nás bude znamenat ztráta informace. První otázka je řešitelná poměrně snadno, neboť se předpokládá, že bude nejlepší se držet obecně platného podvědomí, kde černá barva v masce nepropustí žádnou informaci, naopak bílá barva se chová jako okno, projde tedy veškerá informace na výstup. Co se definice týče, v prvním případě (Implementace masky dostupné v knihovně JImage) bude maska definována dvěma body jež vytvoří obdélník, který bude pro nás znamenat ztrátu informace. Zmíněný postup je jako komponenta GP funkční, leč ne nejvhodnější přístup. Daleko lepší přístup bude předat celou masku na vstupu a s tou pak bude systém dále pracovat. Tohoto postupu je využito pro aplikaci masky na obrazy ve stupni šedi a následně i RGB obrazy. Odpadá tak nutnost definovat nadbytečné parametry a můžeme vytvořit naprosto libovolné tvary masky vrstvy. Podrobnější popis bude uveden v následujících kapitolách. Nyní zůstává otázka ztráty informace. Systém by mělo být možno nakonfigurovat tak, aby v případě potřeby byla ztráta informace reprezentována buď hodnotou #000h nebo #fffh. Záleží na barevném odstínu vstupních dat.

Pro demostraci byla vytvořena komponenta umožňující aplikovat čtvercovou, resp, obdélníkovou masku a definovat barvu pro ztrátu informace. Tu je možné realizovat pomocí tzv. oblasti zájmu neboli *Region of interest - ROI*. Z něj potom můžeme zobrazit i samotnou masku. Náhled funkčnosti je částečně zřejmý z UML diagramu na obrázku 5.8.

Jádrem je třída `MaskFilterJimage` v níž je pomocí dvou bodů definována maska vrstvy. První bod určuje levý horní roh, kde proměnná `rTop` udává vzdálenost od horního okraje okna, proměnná `rLeft` udává vzdálenost od levé stěny okna. Analogicky je definován druhý bod (umístěn diagonálně) pomocí proměnných `rRight` a `rBottom`. Metoda `run` se po té stará o vlastní vyplnění definovaného obsahu patřičnou barvou, přičemž je hlídáno, aby nedošlo k přetečení rozměru okna. I kdyby se tak stalo bez ošetření, vyplní se obsah po daný okraj a dál se již nepokračuje. Pro ověření výsledků slouží ve třídě `main` metoda `runPlugin`, které jsou na vstupu předány všechny nezbytné parametry (vytvořením instance třídy `MaskFilterJimage` a naplnění konkrétních hodnot pro vytvoření masky) a zdrojový obraz. V evolučním systému by se pak měl aplikovat požadovaný filtr na celou oblast vyjma vyznačeného místa.

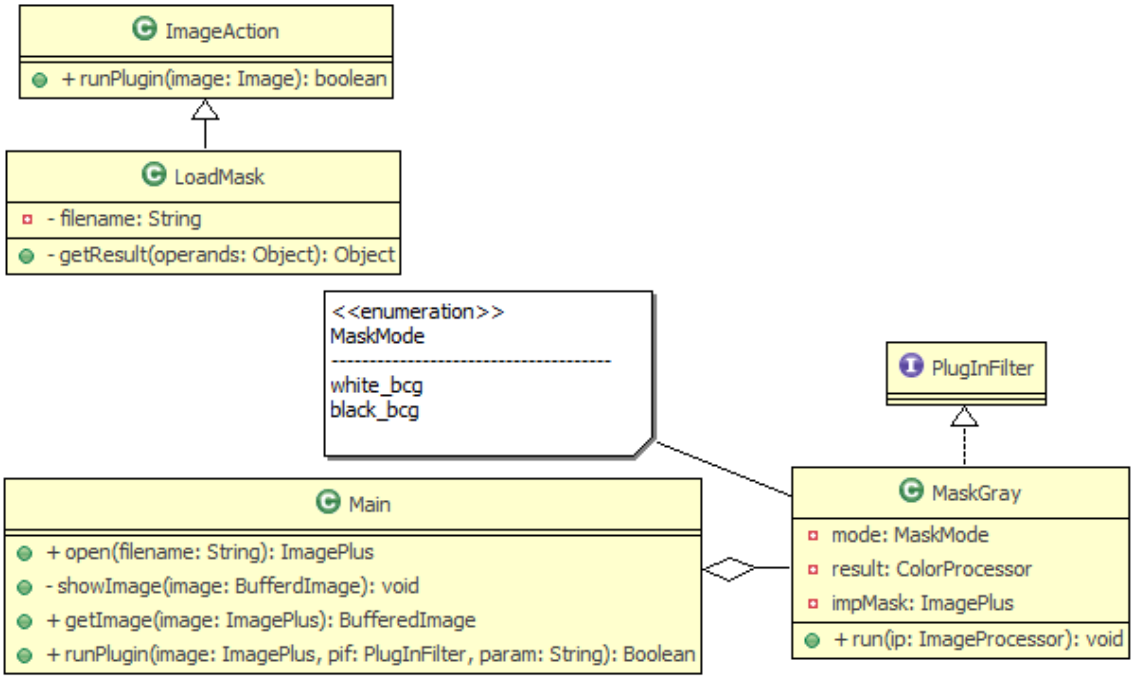


Obrázek 5.8: UML diagram komponenty GP pro implementaci masky vrstvy

5.4 Implementace univerzální masky pro obrazy ve stupních šedi

Jak bylo zmíněno výše, pro evoluční systém je třeba zvolit univerzálnější postup pro definici masky obrazu. Proto byla vytvořena další komponenta s odlišným přístupem. Maska vrstvy je předána jako vstupní parametr. Jedná se o obraz ve formátu *.jpg, *.gif, *.png, který je následně konvertován algoritmem do stupní šedi. Jedině tak můžeme dosáhnout 256 hladin, které pak prakticky specifikují průhlednost. Tím odstraňujeme další z předchozích neduhů, nyní můžeme informaci nejen odstranit, ale také potlačit. Černá barva bude opět znamenat absolutní ztrátu informace, bílá naopak nulovou ztrátu hodnot a např. 50% šedá barva odstraní polovinu informace s ohledem na zvolené pozadí snímku.
Aby bylo možno načítat masku zcela univerzálně byla vytvořena další komponenta `LoadMask`. Jedná se o další třídu starající se o načtení masky dle zadané cesty. Ta je uložena do proměnné `filename`. Při vytvoření objektu typu `LoadMask` je proměnná filename naplněna automaticky kontstruktorem. O samotné předání masky se pak stará metoda `getResult`, kde pomocí specifikované cesty dojde k jejímu načtení a následně je předána na výstup v jako datový typ `ImagePlus`. Výše zmíněné je zřetelné

z obrázku 5.9.



Obrázek 5.9: UML diagram komponenty GP pro načtení masky vrstvy

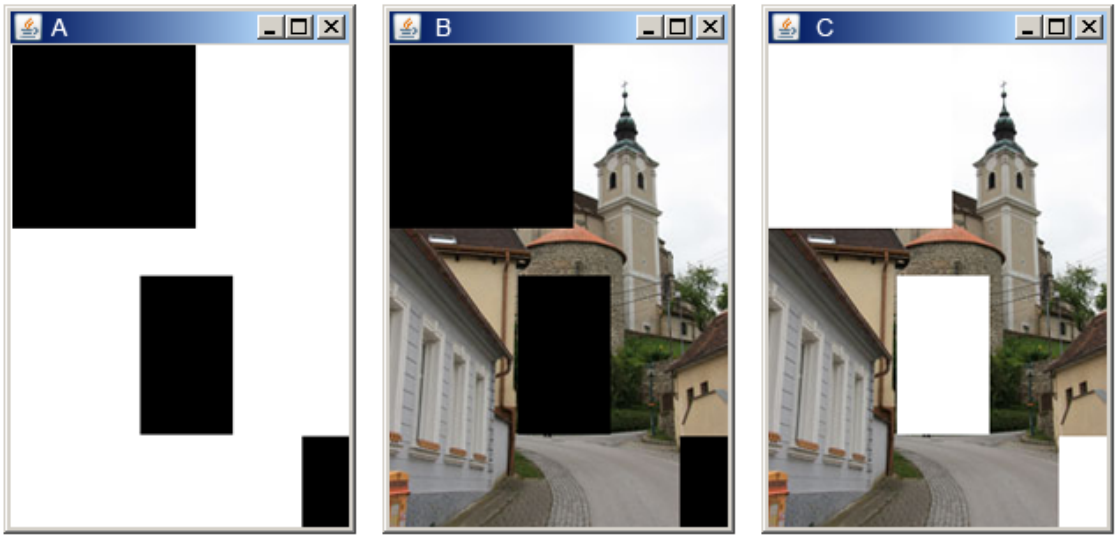
Funkčnost masky co by komponenty pro GP je nejlépe zřetelná z jeho vývojového digramu (příloha A.1). V první řadě jsou inicializovány všechny potřebné proměnné, jsou zjištěny rozměry vstupního obrázku a jsou zpřístupněny jeho pixely pomocí instance `ImageProcessor`. Dále je maska kontertována do 8 bitové šedi a je provedena inverze přes všechny pixely kvůli dalším výpočtům. Dle rozhodnutí o barvě pozadí se volí algoritmus pro aplikaci dané masky na vstupní obraz. Při černém pozadí odečteme od obrazu hodnotu masky. V případě, že je hodnota výsledku menší jak nula, je automaticky výsledná hodnota nulová, v opačném případě je hodnota zachována. Výpočet probíhá přes celou matici hodnot, tedy přes dvourozměrné pole. Výsledek je v každém cyklu ukládán do výstupní matice. Ta je po skočení obou cyklů předána na výstup a jsou aktualizovány všechny body ze vstupního obrazu. Je-li zvoleno bílé pozadí jsou principy výpočtu analogické, mění se pouze jeho logika. Nedochází k odčítání masky ale naopak se sčítá se vstupním obrazem. Nehlídáme tak nulování hodnot, ale naopak sledujeme, zda nebyla překročena hodnota 255, resp. bílá barva. V případě, že ano, je na výstup předána hodnota 255. Výsledkem je v obou případech upravený obraz v nemž jsou požadovaná místa vymazána nebo utlumena do požadované barvy pozadí. Pro evoluční systém je důležitý datový typ výstupu. Ten musí být zachován (`ImagePlus`), aby s ním bylo mažné i nadále pracovat.

Výše uvedený postup je již poměrně univerzální pro celý evoluční systém, nicméně by se mu dal vytknout ještě jeden nedostatek. A to, že pracuje pouze s jedním kanálem, resp. s kanálem šedi. V praxi bude jistě třeba zpracovávat tímto způsobem také barevné obrázky a nejlépe tak, abychom dokázali pracovat s co nejširším spektrem formátů. Proto byla také naprogramována poslední komponenta pro GP jež zmíněný neduh odstraňuje. Pracuje totiž s barevnými obrázky rozličných formátů (*.jpg, *.gif, *.png), resp. se třemi kanály (RGB).

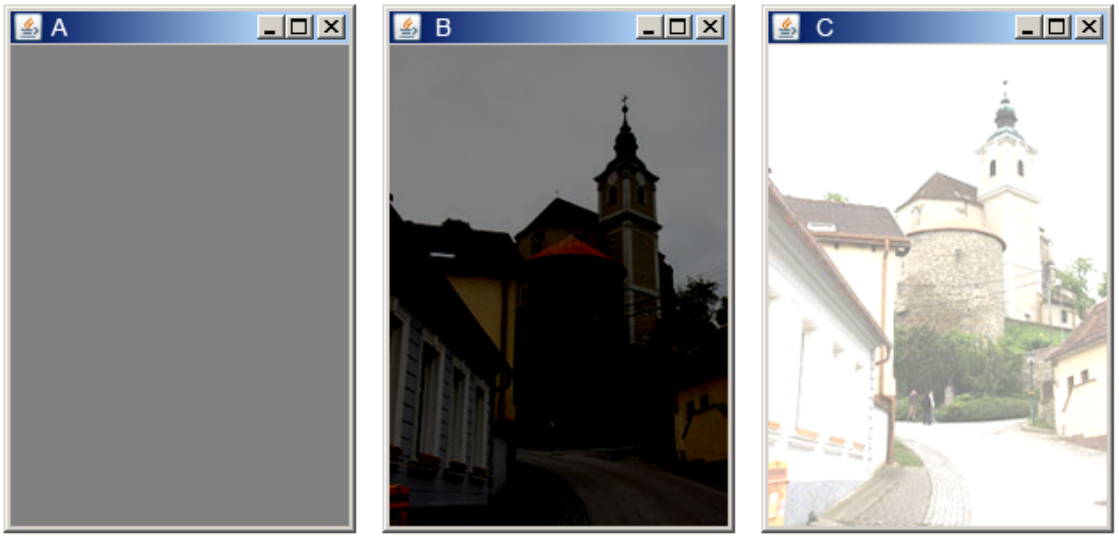
5.5 Implementace univerzální masky pro obrazy v režimu RGB

V této kapitole bude představena poslední komponenta pro GP a mimo jiné budou také prezentovány výstupy po aplikaci masky obrazu na barevné obrázky. Princip se od předchozí metody liší v zásadě v práci ne s jedním, ale se třemi kanály zaráz. Obraz se tak musí rozdělit po vstupu do tří dvou rozměrných polí a masku je pak třeba aplikovat na každý kanál zvláště. Nakonec je třeba všechny upravené vrstvy sjednotit do výstupního obrazu.
Z hlediska objektů je situace velice podobná jako v předchozím případě. Maska je opět předávána vytvořením instance třídy `LoadMask`. O intermpretaci výsledků se opět starají metody `showImage` a `getImage`. Pro evoluční systém je ovšem nejdůležitější valstní komponenta, její vstupy, výstupy a vnitřní algoritmus.
Jedná se o třídu, ketrou jsem nazval `MaskFilterRGB`. Její vnitřní algoritmus je znázorněn v příloze jako obrázek A.2. Na počátku je opět třeba inicializovat nezbytné proměnné, pole a zjistit rozměry vstupního obrazu. Dále je třeba jej rozdělit na jednotlivé barevné kanály a získat přístup k daným pixelům. Totéž je třeba udělat i se vstupující maskou obrazu. Zde je mimo jiné také provést inverzi pixelů pro pozdější matematické výpočty. Následně se algoritmus rozhoduje, zda bude operovat s tmavým (černým), či světlým (bílým) pozadím. V případě černého jsou pro každý pixel v každém kanálu vypočten rozdíl obrazu a dané masky. Pro každý barevný kanál se pak kontroluje zda obdržená výsledná hodnota není záporná. Pakliže není je výsledek předán na výsup. V opačném případě je nastavena nulová hodnota. Zmíněný postup se aplikuje přes dvourozměrné pole všech tří kanálů. Výsledkem jsou tři matice, pro každou barvu jedna, které jsou dále převedy zpět na datový typ `ImageProcessor`. Pro bílé pozadí je princip algoritmu obdobný, mění se ovšem jeho logika. Pro všechny tři kanály se vypočítává součet každého pixelu vstupního obrazu a masky. Pro každou barvu se pak hlídá, zda hodnota výsledku nepřekročila hodnotu 255.

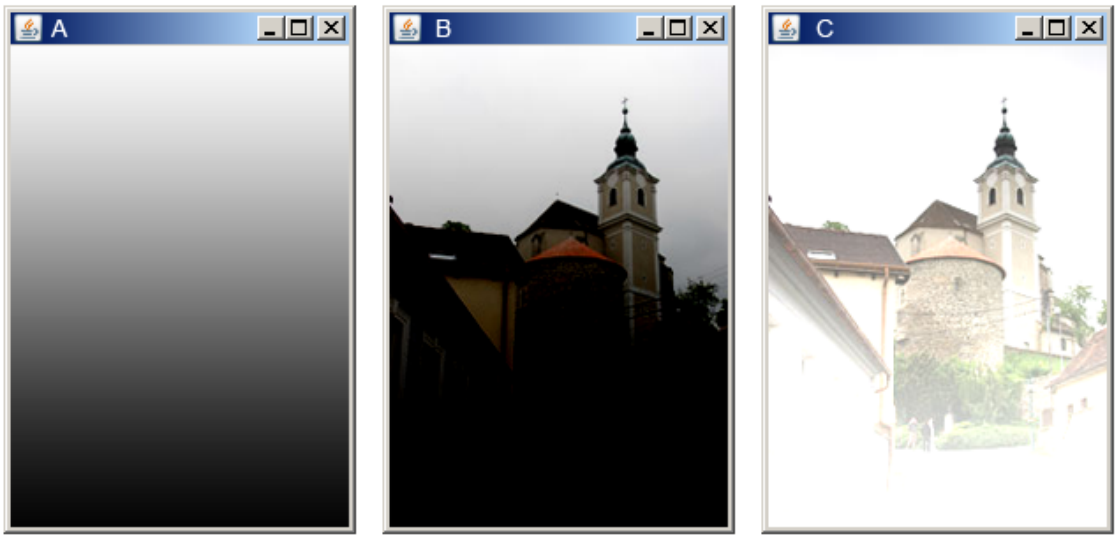
Nasledující ukázky dokumentují některé získané výsledky pomocí výše zmíněné komponenty pro GP.



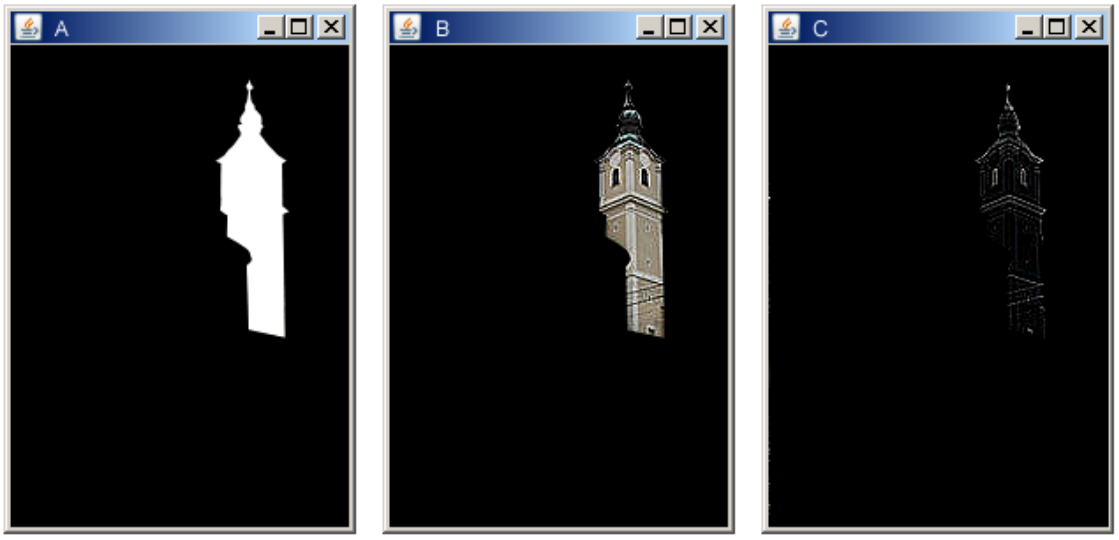
Obrázek 5.10: Aplikace Black&White masky obrazu, obr.A - maska, obr. B - výsledek po aplikaci masky se ztrátou informace do černého pozadí, obr. C - výsledek po aplikaci masky se ztrátou informace do bílého pozadí



Obrázek 5.11: Aplikace 50% šedé masky obrazu, obr.A - 50% šedá maska, obr. B - výsledek po aplikaci masky se ztrátou informace do černého pozadí (průchod 50% informace), obr. C - výsledek po aplikaci masky se ztrátou informace do bílého pozadí (průchod 50% informace)



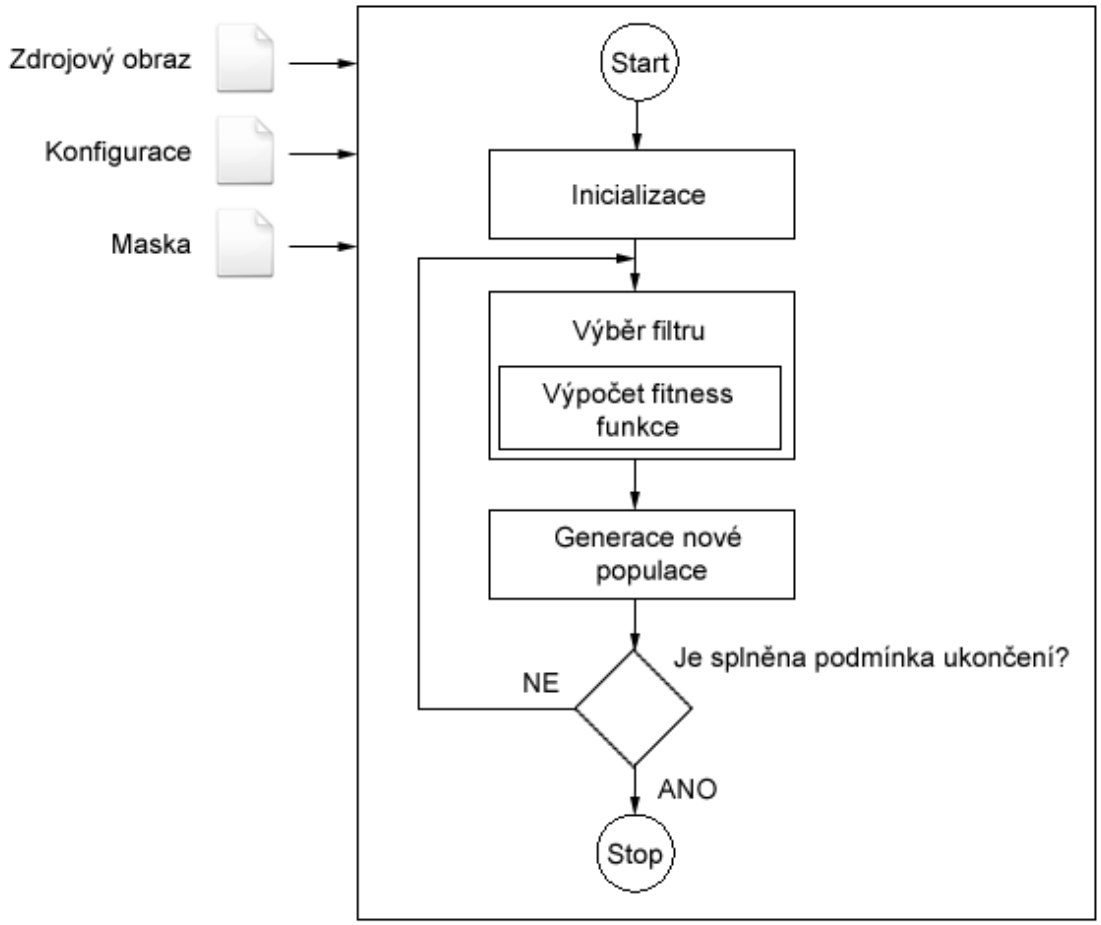
Obrázek 5.12: Aplikace lineární masky obrazu, obr.A - lineární maska, obr. B - výsledek po aplikaci lineární masky se ztrátou informace do černého pozadí (průchod 0 - 100% informace), obr. C - výsledek po aplikaci lineární masky se ztrátou informace do bílého pozadí (průchod 0 - 100% informace)



Obrázek 5.13: obr.A - maska obrazu, obr. B - výsledek po aplikaci zostření pouze na požadovanou oblast, obr. C - aplikace Laplaciánu na požadovanou oblast

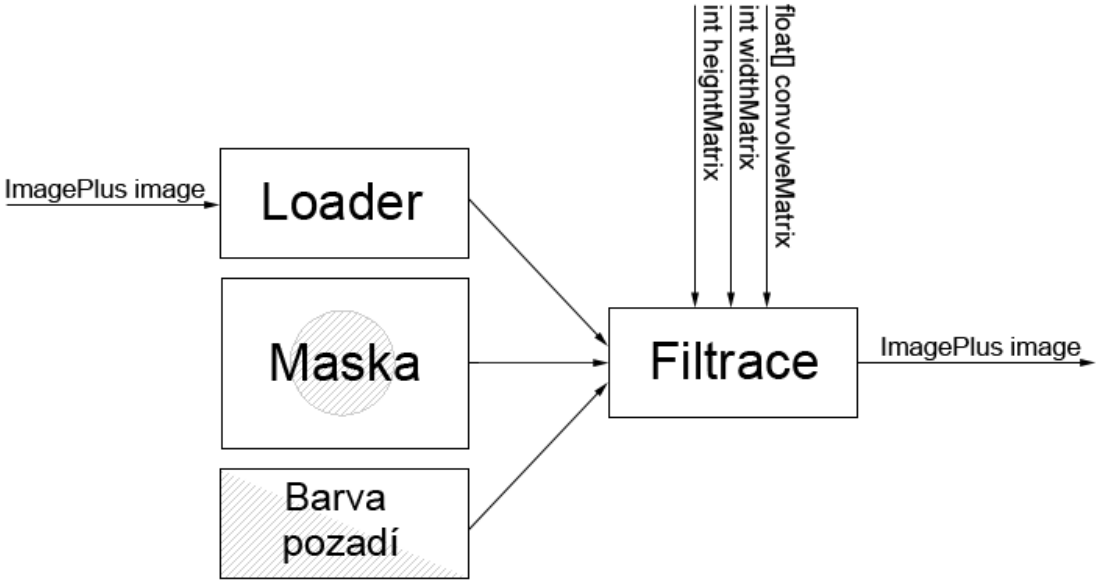
5.6 Implementace vytvořených komponent pro Evoluční Programování

Výše bylo uvedeno šest komponent pro genetické programování, nyní je třeba dát v souvislost vytvořené komponenty, resp. jejich implementaci do evolučního systému. Jedná se o expertní systém, kterému se na vstupu poskytne množina vstupních komponent. Ten s využitím principů evolučního programování bude generovat nové filtry na základě vyhodnocení jejich kvality.
Začlení komponent do systému je částečně zřejmé z obrázku 5.14. Na počátku je



Obrázek 5.14: Evoluční algoritmus - začlenění vytvořených komponent

systém inicializován počáteční konfigurací, je mu předán vstupní obraz, v případě potřeby maska obrazu. Následně může expertní systém začít s návrhem jednotlivých filtrů, ty jsou aplikovány na vstupní obraz a podle předlohy je posuzována relevance použitého filtrů. V případě, že jsou výsledky uspokojující, systém vyhodnotí filtr jako použitelný a může se dále aplikovat na obdobné předlohy.



Obrázek 5.15: Komponenty pro začlenění do evolučního systému

Z programátorského hlediska je nezbytné dodržet vstupní a výstupní formát jednotný. Jako vstupní datový formát je využíván `ImagePlus` specifikovaný knihovnou JImage. Pomocí instance `ImageProcessor`, popř. `ColorProcessor` jsou zpřístupněny jednotlivé pixely a můžeme s nimi dále manipulovat (aplikace filtrů, masek, iverze, matematické operace, apod.). Po provedení všech potřebných operací jejíž za jejichž sled odpovídá samotný evoluční systém, jsou data updatována a ve stejném datovém typu předána na výstup, kde jsou vyhodnocena. Výše uvedené pro názornost reprezentuje obrázek 5.15.

6 ZÁVĚR

Diplomová práce se zabývá návrhem obrazových filtrů pro evoluční programování. Bylo vytvořeno celkem 6 komponent pro evoluční systém vyvíjený skupinou na VUT FEKT Brno. Pro tvorbu zmíněných komponent bylo nezbytné vybrat vhodnou knihovnu pro zpracování obrazu, patřičně ji prozkoumat a získané informace aplikovat při praktickém návrhu komponent pro zmíněný evoluční systém.

V práci byly popsány celkem čtyři knihovny pro zpracování obrazu. Z jejich průzkumu vyplívá, že jako nejvýhodnější z nich se pro další využití jeví knihovna JImage. Mezi její výhody patří fakt, že je postavená na objektové platformě JAVA a je v současnosti poměrně rozšířenou knihovnou pro obrazové zpracování. Proto jsou pro ni nadále vyvíjeny další komponenty a opravovány vzniklé chyby. Zvolená knihovna byla dále v práci podrobně popsána a byly představeny možnosti jejího využití, jak z uživatelského, tak programátorského hlediska. Z pozice programátora můžeme konstatovat, že se jedná o relativně zdařilý a promyšlený projekt. Knihovna logicky odděluje funkční části, což zvyšuje přehlednost při samotném programování aplikací využívajících instance a metody knihovny ImageJ.

Po podrobném průzkumu celé knihovny ImageJ se podařilo zpřístupnit hodnoty jednotlivých pixelů, jak u obrázků černobílých, ve stupních šedi tak i barevných a provádět s nimi potřebné operace. Dvě komponenty pro GP představují možnosti implementace obrazových filtrů. Metoda 2D konvoluce nebyla zvolena náhodně, ale byl brán zřetel na její další využití pro evoluční programování. Máme tedy k dispozici nástroj, pomocí kterého můžeme vytvořit konvoluční filtr a aplikovat jej na patřičný snímek. Toho můžeme s výhodou využít pro evoluční programování, kdy takovéto filtry, případně jejich kombinace, budeme aplikovat na daná obrazová data a následně, dle předloženého vzoru, posuzovat relevantnost provedených úprav pro další využití.

V jistých případech by bylo velice výhodné aplikovat potřebné úpravy jen na určitou část předložených dat. Toho je možné docílit definicí bitové masky, která nám přesně určí v jakých částech obrazu se mají změny provádět a v jakých naopak nemají. Proto vznikly další tři komponenty, které zmíněnou funkci plní. V práci jsou dokoumentovány jejich slabiny, úskalí při nasazení nebo naopak výhody konkrétních řešení. Z výše uvedého lze vyvodit nejlepší uplatnitelnost poslední komponenty jež pracuje s barevnými obrázky, resp. se všemi třemi kanály v součinnosti s další komponentou, která se stará o načítání bitové masky z externího zdroje. Ta nám následně určí konkrétní oblasti pro aplikaci požadovaných úprav.

Celý expertní systém bude zpracovávat veliké množství dat a vzhledem k faktu , že dokážeme zredukovat prováděné úpravy jen na potřebné oblasti, docílíme výrazné úspory jak z hlediska výkonu, tak z hlediska paměťové náročnosti.
Za přínos práce lze považovat prozkoumání knihovny JImage a její využití při tvorbě konkrétních komponent aplikovatelných pro evoluční systém vyvíjený na VUT FEKT Brno. Byla tak prakticky vytvořena spojnice mezi knihovnou JImage a jejím přímým využitím pro evoluční systém.

LITERATURA

[1] BURGET, R. *MTIN - Skriptum Teoretická informatika, 08optimalizace* [pdf]. Skriptum, ústav Teleinformatiky, FEKT VUT, Brno, poslední aktualizace 2.1.2008 [citováno 2009-11-30].

[2] CHEN, S. *Genetic algorithms and genetic programming in computional nance.* Dordrecht: Kluwer Academic Publishers, 2002. 489 s. ISBN 0-7923-7601-3.

[3] OBITKO, M. *Genetic algorithms* [online]. 1998 [cit. 2009-12-01]. Dostupný z: `http://www.obitko.com/tutorials/genetic-algorithms/`.

[4] Jyh-Shing Roger Jang, Chuen-Tsai Sun, Eiji Mizutani. *Neuro-Fuzzy and Soft Computing.* Prentice Hall, 1997. 614 s. ISBN: 0-13-261066-3.

[5] KOZA, John R. *Survey of genetic algoritms and genetic programming. WESCON Conference.* 1995, IEEE. 594 s.

[6] BURGER W., BURGE M., J. *Digital Image Processing: An Algorithmic Introduction using Java.* Springer, 1 edition (November 28, 2007). 566 s. ISBN-10: 1846283795.

[7] ŘÍHA, K. *Pokročilé techniky zpracování obrazu* [pdf]. Skriptum, ústav Teleinformatiky, FEKT VUT, Brno, poslední aktualizace 26.1.2010 [citováno 2010-04-11].

[8] BURGER W., BURGE M., J. *Principles of Digital Image Processing: Fundamental Techniques (Undergraduate Topics in Computer Science)* . Springer, 1 edition (March 26, 2009). 259 s. ISBN-10: 1848001908.

[9] ŠONKA, M., HLAVÁČ, V., BOYLE, R., *Image Processing Analysis, and Machine Vision.* Pacific Grove : Brooks/Cole Publishing Company, 1998, 770 s. ISBN 0-534-95393-X.

[10] HLAVÁČ, V., SEDLÁČEK, M., *Zpracování signálů a obrazů.* Praha : ČVUT, 2002. 220 s. ISBN 80-01-02114-9.

[11] *Jimage - Image Processing and Analysis in Java* [online]. [2009][15-12-2009]. Dostupný z URL: `http://rsbweb.nih.gov/ij/`.

[12] *NeatVision - An Image Analysis Software Development Environment* [online]. [2003][15-12-2009]. Dostupný z URL: `http://neatvision.eeng.dcu.ie/`.

[13] *ImaLab - A Free Experimental System for Image Processing* [online]. Ver. 2.3 [15-12-2009]. Dostupný z URL: `http://imlab.sourceforge.net/`.

[14] *UTHSCSA ImageTool* [online]. Ver. 3.0 [15-12-2009]. Dostupný z URL: `http://ddsdx.uthscsa.edu/dig/itdesc.html`.

[15] BAILER, W. *Writing ImageJ Plugins - A Tutorial* [pdf]. Version 1.71 (July 2, 2006) [cit. 2009-12-09]. Poslední aktualizace 17. 8. 2005. 18 s. Dostupný z URL: `http://cs.joensuu.fi/pages/ageenko/public/notes/ea_imagej.pdf`.

[16] Podlasov, A., Ageenko, E. *Working and Development with ImageJ* [pdf]. Department of Computer Science. University of Joensuu. 2003 [cit. 2009-12-09]. Poslední aktualizace 2. 7. 2006. 57 s. Dostupný z URL: `http://www.imagingbook.com/fileadmin/goodies/ijtutorial/tutorial171.pdf`.

[17] PRŮŠA, Z. *Intel Performance Primitives filtrace, transformace, principy ztrátové komprese* [pdf]. Cvičení do předmětu MGMP, ústav Teleinformatiky, FEKT VUT, Brno. Poslední aktualizace 26.10.2009 [citováno 2009-11-30].

[18] DORAZIL J. *Analýza a segmentace tomografických obrazů*. Brno: Vysoké učení technické. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací, 2009. Počet stran 56 s. Diplomová práce. Vedoucí práce byl Prof. Ing Zdeněk Smékal, CSc.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

API Application Programming Interface

FIFO First In First Out

JRE Java Runtime Environment

JVM Java Virtual Machine

NIH National Institutes of Health

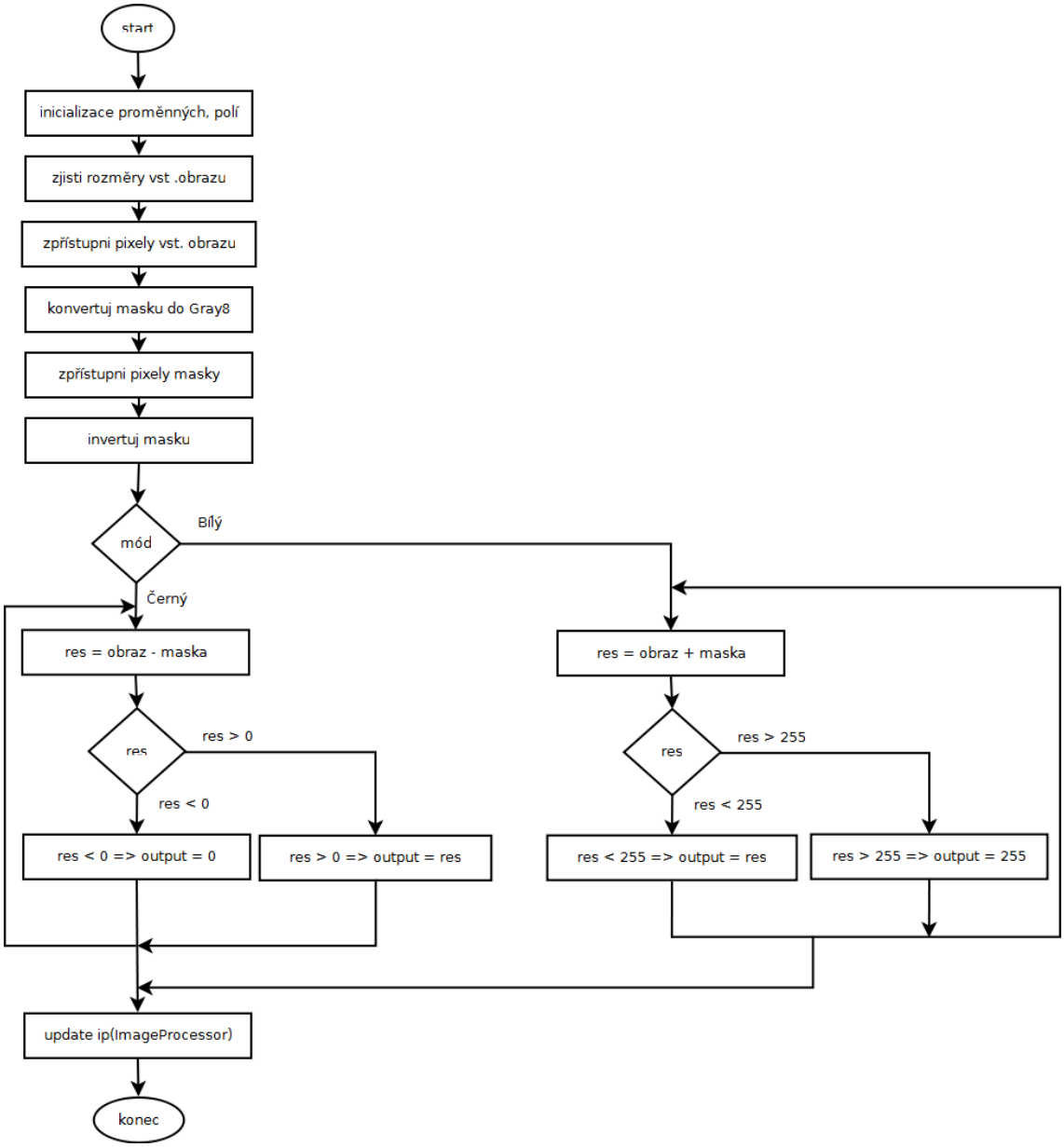
ROI Region Of Interest

SDK Software Development Dit

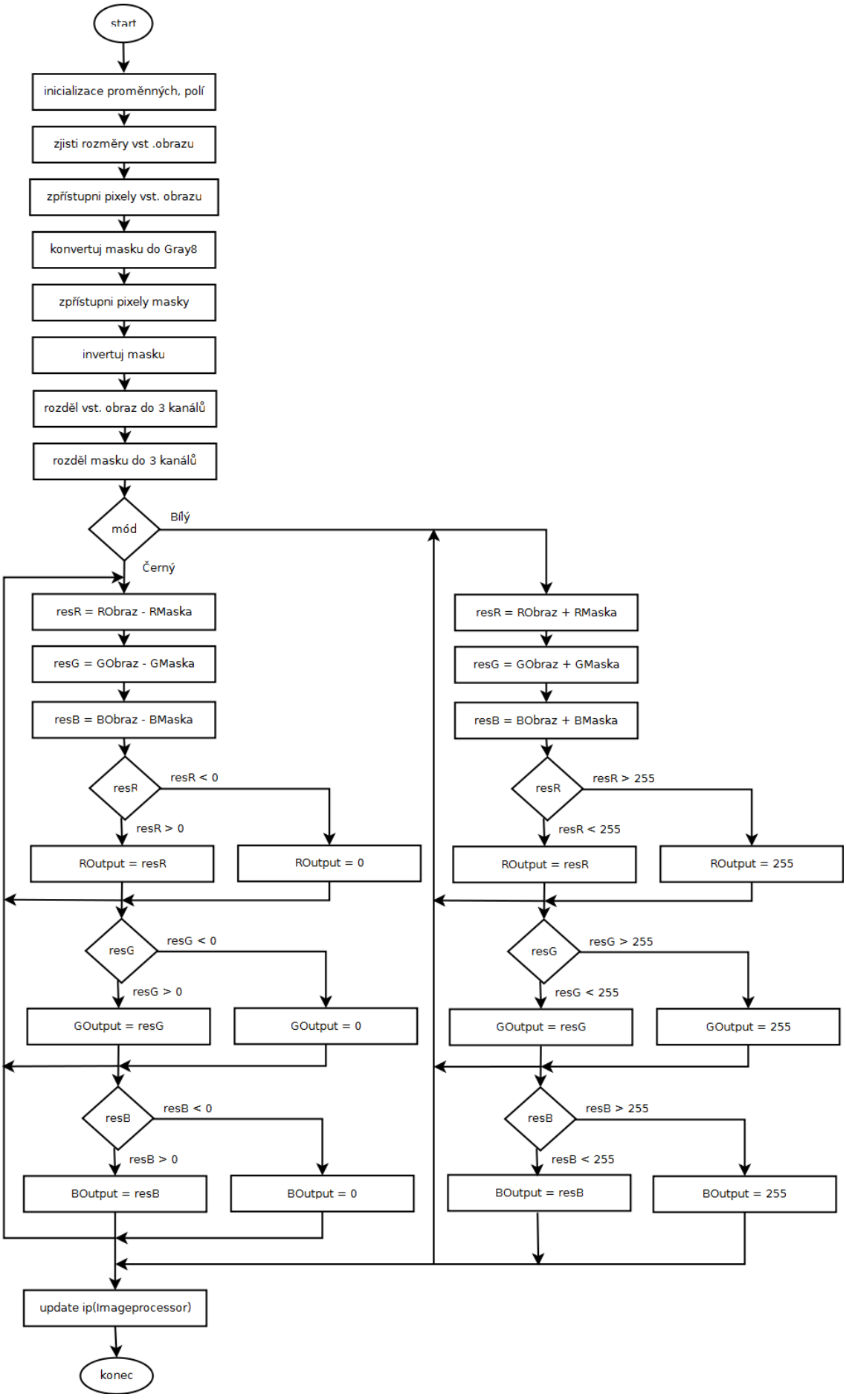
UML Unified Modeling Language

SEZNAM PŘÍLOH

A PŘÍLOHY - VÝVOJOVÉ DIAGRAMY



Obrázek A.1: Algoritmus funkce masky pro obrazy ve stupních šedi



Obrázek A.2: Algoritmus funkce masky pro obrazy v režimu RGB