VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

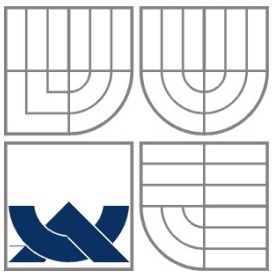
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MORFOLOGICKÉ OPERACE VE ZPRACOVÁNÍ OBRAZU

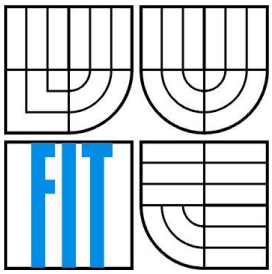
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. MICHAELA KOLOUCHOVÁ
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

MORFOLOGICKÉ OPERACE VE ZPRACOVÁNÍ OBRAZU

MORPHOLOGICAL OPERATIONS IN IMAGE PROCESSING

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. MICHAELA KOLOUCHOVÁ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. ADAM HEROUT, Ph.D.

BRNO 2008

Zadání DP

Licenční smlouva

Morfologické operace ve zpracování obrazu

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracovala samostatně pod vedením Ing. Adama Herouta, Ph.D. Všechny zdroje, prameny a literaturu, které jsem při vypracování používala nebo z nich čerpala, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

……
Michaela Kolouchová
16. května 2008

Poděkování

Ráda bych poděkovala Ing. Adamu Heroutovi, Ph.D. za ochotu a pomoc při vedení této diplomové práce a také svým rodičům za podporu během studia.

Abstrakt

Matematická morfologie vychází z teorie množin a využívá vlastností tzv. bodových množin. Jednu bodovou množinou představuje samotný obraz a druhou (obvykle menší) tzv. strukturní element. Morfologické transformace jsou transformace „z obrazu do obrazu“ založené výhradně na několika základních množinových operátorech. Relace uspořádání mezi obrazy a obrazové transformace hrají v matematické morfologii klíčovou roli. Základní morfologické operace jsou dilatace, eroze a tref či miň. Dalšími operacemi popsanými v následující práci jsou otevření a uzavření. Původně se používaly morfologické operátory pouze pro binární obrazy, postupně však byly zobecněny i pro šedotónové a barevné obrazy. Tato práce popisuje základy morfologie ve zpracování obrazu a některá její praktická použití pro filtrování a segmentaci obrazu.

Klíčová slova

matematická morfologie, bodová množina, strukturní element, morfologická transformace, morfologický operátor, množinové operace, relace uspořádání, dilatace, eroze, tref či miň, otevření, uzavření, binární obraz, šedotónový obraz, barevný obraz , zpracování obrazu, filtrování obrazu, segmentace obrazu

Abstract

Mathematical morphology stems from set theory and it makes use of properties of point sets. The first point set is an origin image and the second one (usually smaller) is a structuring element. Morphological image transformations are image to image transformations based on a few elementary set operators. Fundamental morphologic operations are dilation, erosion and hit or miss. Next operations described in this work are opening and closing. Originally morphological operators were used for binary images only, later they were generalized for grey tone and color ones. This work describes the basic morphological image processing methods including their practical usage in image filtering and segmentation.

Keywords

mathematical morphology, point set, structuring element, morphological transformation, morphological operator, set operators, ordering relations, dilation, erosion, hit or miss, opening, closing, binary image, gray tone image, color image, image processing, image filtering, image segmentation

Michaela Kolouchová: Morfologické operace ve zpracování obrazu, diplomová práce, Brno, FIT VUT v Brně, 2008

Obsah

1 Úvod

1.1 Matematická morfologie

Matematická morfologie se začala vyvíjet v šedesátých letech (konkrétně v roce 1964) a svým matematickým aparátem vycházejícím z algebry nelineárních operací do značné míry při zpracování signálů či obrazů předstihuje tradiční lineární přístup, který využívá lineární kombinaci (*konvoluci*) bodových zdrojů představovaných *Diracovými impulsy*. Jde např. o předzpracování obrazu, o segmentaci s důrazem na tvar hledaných objektů nebo o kvantitativní popis nalezených objektů. Hlavními průkopníky matematické morfologie byli Francouzi *Georges Matheron* a *Jean Serra*.

Operátory matematické morfologie se obvykle používají tam, kde je požadován krátký čas zpracování. S jejich implementací se můžeme setkat v mnoha nástrojích pokročilých softwareových balíků pro zpracování obrazu. Aplikačními oblastmi jsou biologie, materiálový výzkum, geologie, kriminalistika, obrazová inspekce v průmyslu, rozpoznávání znaků a dokumentů, atd. Je přirozené, že morfologické metody lze použít nejen pro 2D obrazy, ale lze je použít i pro zpracování 1D signálů.

1.2 Co je obsahem práce

V první části této práce (2. a 3. kapitola) se čtenář seznámí se základy zpracování obrazu, s možnými reprezentacemi obrazu a s teorií, ze které morfologie nad obrazem vychází. Zde je třeba si uvědomit základní rozdíly mezi binárním, šedotónovým a barevným obrazem v různých barevných modelech, především pak to, čím je reprezentován jeden obrazový bod. Dále jsou definovány základní morfologické pojmy jako *bodová množina*, *strukturní element* a operace *dilatace*, *eroze*, *otevření a uzavření* pro binární, šedotónové a také barevné obrazy. Pro úplnost je přidána kapitola o operaci *tref či miň* a *skeletech*.

Druhá část textu (4. a 5. kapitola) je zaměřená prakticky. Je zde popsáno použití morfologických operací při filtrování a segmentaci obrazu i s konkrétními příklady použití, které byly realizovány v aplikaci vytvořené v rámci této práce. V 5. kapitole je uvedena implementace této aplikace. V příloze lze nalézt manuál pro její použití.

V závěru jsou zhodnoceny dosažené výsledky, přínos práce a návrh možných pokračování .

2 Úvod do zpracování obrazu

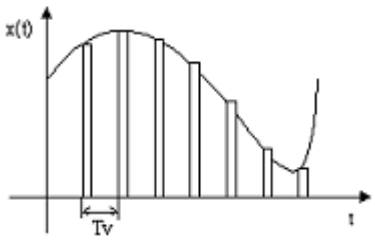
Zpracování obrazu může být chápáno jako zpracování vícerozměrného signálu. Obrazem lze označit obraz na sítnici lidského oka, obrazovku nebo třeba fotografii [2]. Obraz může být modelován matematicky pomocí spojité skalární funkce dvou nebo tří proměnných, která se nazývá *obrazová funkce*. My budeme uvažovat statický obraz popsaný obrazovou funkcí dvou souřadnic *f(x,y)* v rovině. Hodnoty obrazové funkce odpovídají některé měřené fyzikální veličině, např. jasu u obrazu nebo teplotě u termovizní kamery. V počítači pracujeme s digitalizovanými obrazy, kde je obrazová funkce *f(x,y)* představována maticí. Prvky matice jsou obrazové elementy (pixely), jejichž hodnota je úměrná množství světelné energie. Plocha vzorku není nekonečně malá. Z hlediska dalšího zpracování je obrazový element dále nedělitelná nejmenší jednotka.

2.1 Digitalizace

Obraz, který snímáme, má teoreticky nekonečný rozsah obrazových hodnot - je možné ho téměř neomezeně přibližovat nebo vzdalovat. Bude však zobrazen v konečném množství pixelů a s konečným počtem barev. Proces přechodu od spojitého obrazu k diskrétnímu se nazývá *digitalizace* a odehrává se ve dvou nezávislých krocích, kterými jsou *vzorkování* a *kvantování* [2]. Do procesu se někdy řadí také *kódování*.

Vzorkování

Při vzorkování je vstupní analogový signál konvertován na posloupnost veličin vyjádřených okamžitými hodnotami původního signálu [3]. V ideálním případě je signál vzorkován *Diracovým impulsem*, reálně potom obdélníkovou funkcí. Ze spojitého signálu získáme signál diskrétní se spojitou velikostí amplitudy. Vzdálenost mezi vzorky je perioda vzorkovacího signálu $T_s = 1/f_s$. Pro vzorkování je nutné dodržet *Shannon-Kotelnikovův* vzorkovací teorém, který říká, že vzorkovací frekvence f_s musí být nejméně 2x větší než nejvyšší modulační frekvence f_{max}: $f_s > 2 * f_{max}$.



vzorkování obrazu

Obrázek 1 – vzorkování obrazu

Kvantování

Původní amplitudě signálu se přiřadí nejbližší kvantovací úroveň [3]. Rozhraní tvoří rozhodovací úroveň v polovině mezi sousedními kvantovacími úrovněmi. Výšky impulsů se dále nepřenášejí

přesně, ale vždy jen s celou úrovní. Tím se do signálu zanáší chyba, která má vlastnosti šumu a nazývá se *kvantizační šum*.

Kódování

Kvantovaný signál se dále binárně kóduje, tj. jednotlivým kvantovacím úrovním se přiřadí binární číslo – slabika [3]. Protože počet rozlišitelných gradačních stupňů je *m=170*, používá se v televizní technice osmibitové kvantování, což znamená 2^8= 256 odstínů pro každý kanál R,G, B. První bit zleva nese největší informaci a nazývá se MSB (*Most Significiant bit* - nejvýznamnější bit). Poslední bit s nejmenším informačním obsahem se nazývá LSB (*Low Significiant Bit* - nejméně významný bit).

2.2 Diskrétní obrazy

Různorodost diskrétních obrazů spočívá v typu numerické informace, která je vázána k jednomu jejich obrazovému bodu (pixelu) [4]. U binárního a šedotónového obrazu je to *skalár*, zatímco u barevného obrazu je to *vektor* o třech nebo čtyřech položkách.

Binární obrazy

Hodnota pixelu binárního obrazu je buď jedna nebo nula. Hodnota jedna představuje objekt a hodnota nula pozadí, nebo naopak. Binární obraz f je mapování podmnožiny D_f ze Z^n nazývané definiční obor f do dvojice {0,1} [4]:

$$f : D_f \subset Z^N \rightarrow \{0,1\}. \tag{2.1}$$

Šedotónové obrazy

Rozsah hodnot pixelu šedotónového obrazu není omezen na {0,1}, ale je rozšířen na rozsáhlejší množinu kladných celých čísel. Šedotónový obraz f je mapování podmnožiny D_f ze Z^n nazývané definiční obor funkce f do ohraničené množiny kladných celých čísel N_0 [4]:

$$f : D_f \subset Z^n \rightarrow \{0,1,.....,t_{\max}\}, \tag{2.2}$$

kde t_{max} je maximální hodnota použitá pro uložení pixelu obrazu (to je $2^n - 1$ pro pixel kódovaný na n bitech). Někdy se můžeme setkat s mapováním na hodnoty z intervalu <0,1>, kde hodnota nula je pro nejnižší jas (černou barvu) a jedna pro nejvyšší jas (obvykle bílou barvu). Z barevného obrazu lze získat šedotónový výpočtem intenzity jednotlivých pixelů podle vzorce uvedeného v kapitole 2.3 o barevném modelu RGB. Šedotónový obraz se na binární konvertuje pomocí tzv. *prahování*. „Šedotónový“ je obecné označení nejen pro šedotónový obraz, ale i pro skaláry, které se mapují na stejnou množinu N_0 (např. jednotlivé barevné složky RGB barevného obrazu).

Barevné (vícekanálové) obrazy

Vícekanálový obraz se skládá z pole jednokanálových (buď binárních nebo šedotónových) obrazů. Všechny pixely obrazu jsou reprezentovány vektorem skalárních hodnot. Dimenze tohoto vektoru je dána počtem dostupných kanálů. Mějme barevný obraz f [4]:

$$f(p) = (f_1(p), f_2(p), \ldots, f_m(p)), \tag{2.3}$$

který má m kanálů (m-dimensionální vektor) a p představuje všechny pixely obrazu. Každý kanál f_i obrazu je obvykle zpracováván jako jeden šedotónový obraz, nezávisle na ostatních kanálech. Existuje mnoho druhů vícekanálových obrazů závisejících na typu informace vztahující se k jednomu pixelu obrazu. Nejčastěji se je možné se setkat s tříkanálovým nebo čtyřkanálovým barevným obrazem. Jeden kanál pro každou ze tří základních barev v barevném modelu (např. v RGB, HSV, HLS, CMY..) a případně čtvrtý tzv. *alfa kanál*, který udává hodnotu průhlednosti. Většinou bývá jeden kanál uložen na 8 nebo 16 bitech, nebo se také používá indexování barev pomocí palety, kde je celý pixel (všechny 3 kanály) na 8 nebo 16 bitech.

Obrázek 2 – rozklad RGB obrazu na jednotlivé („šedotónové“) kanály

Počet kanálů muže být někdy hodně vysoký, třeba u multispektrálních nebo hyperspektrálních obrázků produkovaných obrazovým spektrometrem. Pak každý kanál odpovídá rozsahu vlnových délek a obsahuje spektroskopické informace. V některých situacích může být vícekanálový obraz složený z korespondujících obrazů stejného objektu získaných různými senzory.

2.3 Barevné prostory a modely

Všechny barvy, se kterými se můžeme v obrazu setkat, jsou tvořeny kombinaci tří (nebo i čtyř) veličin, které popisují barevný prostor. Soubor základních barev, pravidla jejich míchání a měněné barevné charakteristiky jsou definovány pomocí barevných modelů**.** Tvorbou barevných modelů (uspořádáním barev) se v minulosti zabývala řada osobností vědy - *Aristotelés, Isaac Newton, Johann Heinrich Lambert, Johann Wolfgang Goethe, James Clerk Maxwell* a další [5]. Tři základní barevné modely jsou *RGB(α), CMY(K)* a *HSV*.

RGB, RGBA

U barevného prostoru **RGB** představují tyto tři veličiny červenou (*Red*), zelenou (*Green*) a modrou (*Blue*) barvu. Ty tvoří vektor o třech složkách z intervalu <0,1>, případně v celočíselném rozsahu 0 (barva není zastoupena) až 255 (největší intenzita barvy), což odpovídá kódování každé ze složek **RGB** v jednom bytu [2]. Nejběžnější kódování je 3 byty na pixel, ale používají se i jiná. Složením 3 složek v nejvyšší intenzitě dostaneme bílou barvu (aditivní skládání barev), v nejnižší barvu černou. Lidské oko vnímá různým způsobem intenzitu jednotlivých barevných složek, pro výpočet jasu se tedy používá empirický vztah:

$$I = 0.299R + 0{,}587G + 0{,}114B. \tag{2.4}$$

Můžeme se setkat i se zkratkou **RGBA** (**RGBα**), která se používá k vyjádření skutečnosti, že barevný obraz zapsaný v **RGB** prostoru je doplněn informací o průhlednosti (*alfa kanál*).

CMY, CMYK

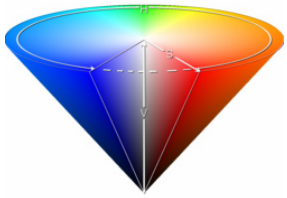
Barevný prostor **CMY-K** (*Kyan* - azurová, *Magenta* - purpurová, *Yellow* - žlutá, *blacK* - černá) představuje subtraktivní míchaní barev. Pokud mají 3 složky z intervalu <0,1> hodnotu nula, pak je výsledná barva bílá. Pokud mají složky hodnotu jedna, tak je výsledkem černá (většinou je to při použití barviv spíše tmavě šedá, proto se u tiskáren přidává ještě černá barva).

Obrázek 3 - rozklad CMYK obrazu na jednotlivé kanály

HSV, HLS

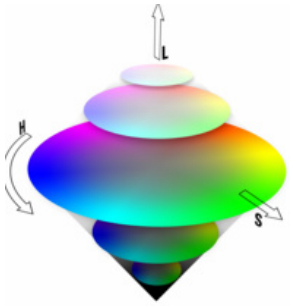
HSV (*Hue, Saturation, Value*) model je známý také jako **HSB** (*Hue, Saturation, Brightness*). Vytvořil ho v roce 1978 Alvy Ray Smith a je to barevný model, který nejvíce odpovídá lidskému vnímání barev [6]. Skládá se ze tří složek (nejsou to základní barvy):

- *Hue* představuje převládající barevný tón (odstín). Udává barvu odraženou nebo procházející objektem. Měří se jako poloha na standardním barevném kole (0° až 360°). Obecně se odstín označuje názvem barvy.
- *Saturation* udává sytost barvy, někdy také označovanou chroma. Určuje čistotu barvy (příměs jiných barev). Představuje množství šedi v poměru k odstínu, měří se v procentech od 0% (šedá) do 100% (plně sytá barva). Na barevném kole vzrůstá sytost od středu k okrajům.
- *Value* vyjadřuje hodnotu jasu danou množstvím bílého světla (relativní světlost nebo tmavost barvy). Měří se také v procentech.



Obrázek 4 - kuželovité znázornění barevného modelu HSV

HLS (*Hue, Lightness, Saturation*) je obdobou prostoru HSV, v němž byl jehlan nahrazen dvojicí kuželů. Tvar odpovídá skutečnosti, že nejvíce různých barev vnímáme při „průměrné“ světlosti (oblast podstav). Schopnost rozlišit barvy klesá jak při velkém ztmavení tak při přesvětlení (oblast obou vrcholů kuželů).



Obrázek 5 – znázornění barevného modelu HLS

Prostory **HSV** a **HLS** umožňují postupně měnit barevné charakteristiky při zachování ostatních typických vlastností barvy [2]. To je příjemné pro uživatele, kteří chtějí definovat barvy přirozenými pojmy, jako sytost, světlost a dominantní barva. Avšak je nutné hlídat hodnoty, protože některé kombinace jsou nesmyslné.

Některé další barevné modely

CIE

Tyto modely jsou definované organizací Commission Internationale de l'Eclairage (The International Commission on Illumination, Mezinárodní organizace pro osvětlení, *CIE*), která byla založena v roce 1931 [5]. Barevné prostory definované *CIE* jsou „nezávislými na zařízení“, neboť označení jednotlivých barevných odstínů nezávisí na subjektivních vlastnostech pozorovatele - proto byl vytvořen tzv. standardní pozorovatel (spíše standardní podmínky pozorování barev). Základem barevných modelů *CIE* jsou *chromatické diagramy*. Patří mezi ně *CIE 1931 (x,y)*, *CIE 1976 UCS (u'v')*, *CIE-UVW*, *CIE L*C*h°*,*CIE-LAB* (*CIE L*a*b**, *CIELAB*) nebo *CIE L*u*v** (*CIE-uv*).

CIE XYZ

Tento model, známý také jako *CIE 1931*, byl prvním matematicky definovaným barevným modelem vytvořeným *CIE*. V rámci tohoto modelu se setkáváme s pojmem barevný (*chromaticity*) diagram. Aby se dala zkombinovat libovolná barva (spektrální křivka) vnímaná lidským okem, tak *CIE* definovala imaginární barvy (*spektrální křivky*) "červená (*red*)", "zelená (*green*)" a "modrá (*blue*)", jejichž kombinací lze přibližně obsáhnout celý rozsah lidského barevného vidění [5].

YC_BC_R

YCbCr nebo *Y'CbCr* patří do rodiny barevného modelu používá se u videa a nebo u digitální fotografie. *Y'* je řazen do luminance (jasu) komponentu a *Cb* s *Cr* jsou modrý a červený chrominanční komponent. Náplní *Y* je odlišení luminance. *Y'CbCr* není absolutní barevný model. Je to způsob kódování *RGB* informací. Přímé zobrazení barev závisí na aktuálním užití barev *RGB* v signálu.

YUV

Jde o barevný prostor používaný v televizním vysílání v normě PAL [5]. Model k popisu barvy používá tříprvkový vektor [Y,U,V], kde *Y* je jasová složka, *U* a *V* jsou barevné složky. *U* je také někdy označováno jako *B-Y* a *V* odpovídá *R-Y*. Barevné složky se používají v rozsahu <-0,5 ; 0,5>, jasová složka má rozsah <0,1>.

NCS (Natural Color System)

Jedná se o systém přirozených barev vytvořený ve Švédsku. Znázorňuje šedé odstíny a odstíny s příměsemi bílé a černé barvy. Základními parametry jsou barevný tón, příměs bílé a černé barvy.

Munsellův barevný prostor

Je to systém třídění barev, zohledňující lidské vnímání vytvořený v roce 1905 Albertem H. Munsellem [12]. Popisuje barvu třemi základními vlastnostmi: odstínem, jasem a sytostí. S jistými modifikacemi se tento prostor používá i dnes, v době počítačů.

2.4 Obrazové transformace

Morfologické transformace jsou transformace „z obrazu do obrazu“. Transformovaný obraz má stejný definiční obor jako vstupní obraz a je mapováním tohoto definičního oboru na množinu nezáporných celých čísel. Pro takové mapování používáme notaci Ψ. V závislosti na tom, jestli je výstupní hodnota daného obrazového bodu určena hodnotou tohoto pixelu, nebo vyžaduje znalost okolních pixelů ve vstupním obrazu, rozlišujeme *bodové* a *lokální transformace obrazu* [4]. Dále se pak můžeme setkat ještě s *transformací globální, vyššího stupně a objektovou*.

Bodová transformace

Výstupní hodnota *value(s)* pixelu *p* je funkcí vstupní hodnoty *value(s)* tohoto pixelu, bez jakéhokoli použití hodnot *value(s)* okolních pixelů [4]. Např. každému obrazovému elementu přiřazuje novou hodnotu stupně šedi jako funkci stupně šedi původního $g = T(f)$, kde f je původní stupeň šedi a g je nový stupeň šedi. Mezi bodové transformace patří různé *jasové operace, zvýšení kontrastu, prahování* (převod šedého obrazu na binární), *adaptivní prahování, negace* (inverze stupně šedi), *operace mezi obrazem a číslem* nebo *operace mezi dvěma obrazy* (logické, tj. průnik, sjednocení, atd.).

Obrázek 6 – barevný obraz konvertovaný na šedotónový a dále pak na binární obraz

Lokální transformace

Narozdíl od bodových transformací výstupní hodnota lokální obrazové transformace daného pixelu je funkcí hodnot pixelů spadajících do určitého okolí daného obrazového bodu. Patří sem většina morfologických operací. Důležitou (nemorfologickou) transformací je dvourozměrná diskrétní *konvoluce* označovaná operátorem *. Konvoluce dvou funkcí je definována předpisem [4]:

$$I'(x,y) = I(x,y) * h(x,y) = \sum_{i=-k}^{k} \sum_{j=-k}^{k} I(i,j).h(x-i, y-j), \qquad (2.5)$$

kde $I(x,y)$ je obraz, se kterým pracujeme a $h(x,y)$ je *konvoluční jádro*. Konvoluční jádro je obyčejně definováno tabulkou a udává, jak se bude náš obraz měnit. Konvoluce je tedy operace, která na každý bod funkce $I(x,y)$ položí konvoluční jádro a získá výstupní funkci $I'(x,y)$. Poté konvoluční jádro posuneme na další pixel a výpočet opakujeme. Výše uvedený postup je základem naprosté většiny

operací s diskrétním obrazem, mezi něž řadíme např. odstraňování šumu, detekci hran nebo vytlačený vzor. V praxi se jedná o pouhou filtraci obrazu za pomocí konvolučního jádra.

Globální transformace

Globální transformace používá k výpočtu hodnoty výstupního obrazového bodu hodnoty všech pixelů vstupního obrazu, tedy nová hodnota elementu je funkcí celého obrazu. Patří sem například histogram hodnot intenzity nebo Fourierova transformace. Právě kvůli globálním transformacím jsou systémy pro počítačové vidění značně pomalé.

Objektové transformace

Většina aplikací počítačového vidění vyžaduje vlastnosti vypočtené na objektové úrovni. Musejí být nalezeny vlastnosti jako velikost objektu, průměrná intenzita, tvar a další charakteristiky, aby nedocházelo k chybným detekcím [13].

2.5 Množinové operátory používané nad obrazy a další pojmy

Základní morfologické operace jsou založeny výhradně na kombinaci množinových operátorů *průniku, sjednocení, doplňku* a *translace* [4]. V matematické morfologii má důležitou roli i *relace uspořádání* a můžeme se setkat také s pojmy jako jsou *komutativita* a *asociativita* nebo *dualita* operací.

Množinové operátory používané nad obrazy

Základními a dalšími důležitými množinovými operátory jsou *sjednocení* $\cup$, *průnik* $\cap$, *doplněk, množinový rozdíl, translace a transponovaná množina*.

Sjednocením dvou binárních obrazů $A \cup B$ je binární obraz, který má hodnotu jedna u všech pixelů, které měly hodnotu jedna u pixelů obrazu *A* nebo *B* (nebo u obou).

$$A \cup B = \{p \,/\, p \in A \, or \, p \in B\} \tag{2.6}$$

Pro šedotónové obrazy sjednocení znamená *point wise maximum* (maximální kritérium ve fuzzy) operátor. Point-wise maximum $\vee$ mezi dvěma obrazy *f* a *g* je definováno pro každý bod *x* následovně [4]:

$$(f \vee g)(x) = \max[f(x), g(x)]. \tag{2.7}$$

Tento operátor můžeme reprezentovat také jako term sjednocení podgrafů [4]:

$$SG(f \vee g) = SG(f) \cup SG(g). \tag{2.8}$$

Průnikem dvou binárních obrazů $A \cap B$ je binární obraz , který má hodnotu jedna u všech pixelů, které měly hodnotu jedna u pixelů obrazu *A* a zároveň u obrazu *B*.

$$A \cap B = \{p \,/\, p \in A \text{ and } p \in B\} \tag{2.9}$$

Šedotónový průnik je nahrazen operátorem *point-wise minimum* (minimální kritérium) $\wedge$ majícím identický definiční obor jako operátor point wise maximum. Point-wise minimum $\wedge$ mezi dvěma obrazy *f* a *g* je definováno:

$$(f \wedge g)(x) = \min[f(x), g(x)]. \tag{2.10}$$

Operátor lze opět reprezentovat také jako term průniku podgrafů [4]:

$$SG(f \wedge g) = SG(f) \cap SG(g). \tag{2.11}$$

Dalším základním operátorem je *doplněk*. Doplněk obrazu *f*, označovaný f^c , je definován pro každý pixel *x* jako maximální hodnota datového typu použitého k uložení obrázku zmenšená o hodnotu obrázku *f* na pozici *x* [4]:

$$f^c(x) = t_{\max} - f(x). \tag{2.12}$$

Obrázek 7 – původní binární a šedotónový obraz a jeho komplement

Množinový rozdíl mezi dvěma množinami *X* a *Y*, označovaný *X\Y*, je definován jako průnik množiny *X* a doplňku množiny *Y* (lze použít pouze pro binární obrazy, pro šedotónové obrazy je množinový rozdíl realizován prostým odečtením hodnot odpovídajících si pixelů):

$$X \setminus Y = X \cap Y^c. \tag{2.13}$$

Translace (posunutí) obrazu *f* ve směru vektoru *b*, se značí f_b. Hodnota posunutého obrazu na daném pixelu *x* je rovna hodnotě původního obrazu na pozici posunuté ve směru opačném, než je vektor posunutí:

$$f_b(x) = f(x - b). \tag{2.14}$$

Nechť $\check{A}$ označuje *symetrickou množinu* (někdy je také označována jako transponovaná množina vůči reprezentativnímu bodu), pak je $\check{A}$ definována takto:

$$\check{A} = \{-a : a \in A\}. \tag{2.15}$$

Relace uspořádání

Výstupní hodnoty základních morfologických operátorů daného obrazového bodu vyžadují uspořádání hodnot vstupního obrazu spadajících do okolí tohoto pixelu. Existují dva druhy uspořádání [4].

Prvním je *částečně uspořádaná množina*, která má relaci '≤' definovanou pro některé členy A a B a splňující tři podmínky:

1. $A \leq A$ (*reflexivita*);
2. $A \leq B$ a zároveň $B \leq A$ jenom když $A = B$ (*anti-symetrie*);
3. pokud $A \leq B$ a zároveň $B \leq C$ potom $A \leq C$ (*tranzitivita*).

Druhým uspořádáním je *úplně uspořádaná množina*, která je částečně uspořádanou množinou splňující následující mocnější formu anti-symetrické vlastnosti nazývanou *trichotomická*: „Pro jakékoli dva prvky A a B musí platit jen jedno z následujících: $A < B$, $A = B$, $A > B$."

Reprezentace pixelů a jejich uspořádání

Pro binární obraz je hodnota pixelu nula nebo jedna, takže při práci s ním jen testujeme do jaké množiny patří. U šedotónového obrazu je přirozeně úplná relace uspořádání a je jasně definována unikátní maximální a minimální hodnota pixelu. Lze zde snadno aplikovat operátor point wise maximum nebo point wise minimum. Co se týče barevných obrázků, které se skládají z více kanálů a jeden jejich pixel je reprezentován vektorem, je jejich porovnávání složitější, protože nad nimi není definováno uspořádání. Toto se řeší různými způsoby, které budou popsány v následující kapitole.

Komutativní a asociativní operace

Nechť M je množina. Zobrazení $*$ z $M \times M$ do M se nazývá *binární operace na množině M* [14]. Operace $*$ může mít různé vlastnosti. Řekněme, že $*$ je *komutativní* operace, pokud pro každé x, $y \in M$ je $x * y = y * x$. Nejznámější příklady komutativní binární operace jsou sčítání $(x + y)$ a násobení $(x \cdot y)$ reálných čísel. Dalšími příklady jsou průnik a sjednocení množin nebo operace maximum a minimum.

Operace $*$ je *asociativní*, pokud pro všechny $x, y, z \in M$ platí $(x * y) * z = x * (y * z)$. Nejznámější příklady asociativních binárních operací jsou opět sčítání a násobení reálných čísel. Dalšími asociativními binárními operacemi jsou třeba průnik a sjednocení množin nebo operace maximum a minimum [14].

Dualita

V matematice má *dualita* numerický význam. Obecně duality převádějí prostým způsobem koncepty, teorémy nebo matematické struktury do dalších konceptů, teorémů nebo matematických struktur [15]. Charakteristickou duální operací je operace umocnění: „pokud duálem *A* je *B*, pak duálem *B* je *A*“. Stejně jako umocnění mají někdy pevné body, tak duálem *A* je někdy *A* samo osobě.

V matematické morfologii se setkáváme s *geometrickou dualitou*. V jedné skupině dualit, koncepty a teorémy určitých matematických teorií jsou mechanicky převedeny do dalších konceptů a teorémů té stejné teorie.Typickým zástupcem je dualita v projekční geometrii. Dalšími příklady jsou dualita polyhedronu, duální graf planárního grafu, duální problém v optimalizační teorii a *De Morganova dualita* v logice. Můžeme setkat i s dalšími dualitami, jako třeba s kontravariantní nebo analytickou.

Homeomorfismus

Homeomorfismus (z řeckého *homeos* = stejný, *morphe* = tvar) je zobrazení mezi topologickými prostory, které zachovává topologické vlastnosti a je *izomorfismem*. Existuje-li mezi dvěma topologickými prostory homeomorfismus, říkáme, že jsou homeomorfní [16].

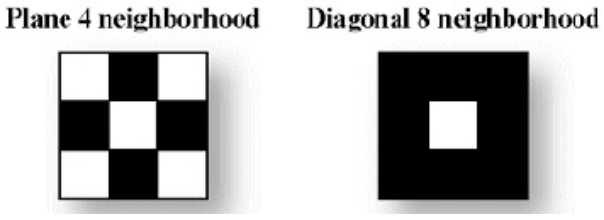
Zobrazení $f: X \rightarrow Y$ nazveme homeomorfismus, pokud

- je prosté
- je spojité
- inverze $f^{-1}: f(Y) \rightarrow X$ je spojitá.

Pokud existuje homeomorfismus *X* na *Y*, nazýváme prostory *X* a *Y* homeomorfní. Homeomorfismy jsou ekvivalence na třídách topologických prostorů [17].

Okolí a sousednost

Pixel v obraze má kolem sebe několik dalších pixelů. V obraze reprezentovaném čtvercovou mřížkou má pixel hranice se čtyřmi pixely a sdílí rohy s dalšími čtyřmi pixely [13]. Říkáme, že pixel je ve *4-okolí (4-sousedství)* dalšího pixelu, pokud spolu sdílí společnou hranici a v *8-okolí (8-sousedství),* když sdílí alespoň jeden roh.



Obrázek 8 – 4-okolí a 8-okolí pixelu

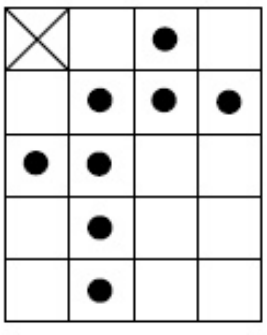
3 Základní morfologické pojmy a operace

Mezi základní morfologické operace patří *dilatace, eroze* a od nich odvozené *otevření a uzavření*. Dále sem lze také řadit transformaci *tref či miň* (hit or miss), která je morfologickým operátorem indikujícím shodu strukturního elementu a části obrazu.

3.1 Základní morfologické pojmy

Matematická morfologie využívá vlastností bodových množin, výsledky z integrální geometrie a topologie. Morfologické operátory jsou lokální nelineární operátory, užívající masky (tzv. *strukturní element*) [1].

Bodová množina



Obrázek 9 – bodová množina

Obrázky lze modelovat pomocí bodových množin libovolné dimenze. Euklidovský prostor E_2 a systém jeho podmnožin je přirozeným definičním oborem pro popis rovinných útvarů. Pro binární obraz je bodovou množinou množina dvojic celých čísel Z_2 a pro šedotónový obraz je to množina trojic Z_3 [7].

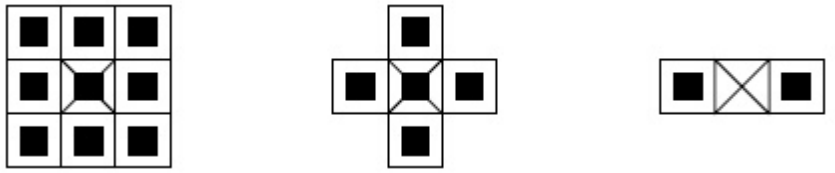
Např.: X={(0,2),(1,1),(1,2),(1,3),(2,0),(2,1),(3,1),(4,1)}

počátek (0,0) 

Morfologická transformace a strukturní element

Morfologická transformace je relace s jinou, menší bodovou množinou B používanou k prozkoumávání obrazu a nazývanou *strukturní element (SE).* Pro obraz představovaný n-dimensionální bodovou množinu je možné použít až n+1-dimensionální strukturní element. Nicméně je dobré vyhnout se pomíchání prostorových složek definičního oboru obrazu se složkami intenzit šedotónové úrovně, a proto je doporučováno používat n-dimensionální strukturní elementy. Tyto strukturní elementy jsou často nazývány *rovinnými (flat),* protože mají jen dvě dimenze (v případě 2-dimenzionálních obrazů). Podobně jsou n+1-dimenzionální strukturní elementy označovány jako *objemové (volumic), nerovinné (nonflat)* nebo *šedotónové* strukturní elementy [4]. Základní morfologické operátory vyžadují znalost počátku SE. Tento počátek umožňuje umístění SE na daném bodě nebo pixelu. Strukturní elementy, které mají stejné vlastnosti pro různé směry, nazýváme *izotropické*.

Morfologickou transformaci si představíme, jako bychom pohybovali strukturním elementem *B* systematicky po celém obraze *X*. Bod obrazu, který se shoduje s počátkem souřadnic SE, nazýváme *okamžitý bod*. Výsledek relace mezi obrazem a SE zapíšeme do okamžitého bodu obrazu.



Obrázek 10 - základní rovinné izotropické strukturní elementy

Ke každé morfologické operaci $\psi(X)$ existuje duální transformace $\psi^*(X)$ vyplývající z množinového doplňku, pro kterou platí [4]:

$$\psi(X) = (\psi^*(X^C))^C. \tag{3.1}$$

Některé morfologické operace, operátory a použití morfologie

- eroze, dilatace (erosion, dilation)
- otevření, uzavření (opening, closing)
- tref či miň (hit or miss), hit and miss
- ztenčování a ztlušťování (thinning, thickening)
- transformace vrchní části klobouku (top-hat transformation)
- fit-and-miss, bottom-hats
- skelet (skeleton)
- morfologický gradient (morphological gradient)
- morfologický Laplacián (morphological Laplacian)
- granulometrie (granulometry)
- geodetické transformace (geodesic transformations)
- geodetické metriky (geodesic metrics)
- morfologické rekonstrukce (morphological reconstruction)
- vzdálenostní transformace (distance function)
- boolean konvoluce (boolean convolution)
- propagace (propagation)
- morfologická segmentace (morphological segmentation)
- morfologická filtrace (morphological filtering)
- analýza textury (texture analysis)
- klasifikace (classification)

3.2 Binární morfologie

Základní jednotkou obrazové informace v morfologickém přístupu je binární obraz s definičním oborem Z_2 a s oborem hodnot {1,0} [1]. S binárním obrazem pracuje tzv. *binární matematická morfologie*. Pro vyjádření morfologických operací nad binárním obrázkem je použit *Minkowského formalismus*. Alternativou by byl *Serrův formalismus*.

Binární dilatace

Jednou ze dvou základních morfologických operací nad binárním obrazem je *dilatace.* Formuluje se otázkou (odpověď pro každý bod strukturního elementu o pozici počátku *x = (i,k)*) [9]:

Kryje strukturní element B o počátku v x některý pixel objektu v f?

Dilatace množiny *X* strukturním elementem *B* se označuje δ_B a je definována jako lokus bodů *x* takových, že *B* zasáhne *X*, pokud je počátek umístěn v *x* [4]:

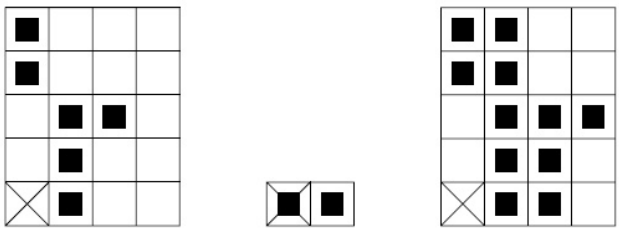
$$\delta_B(X) = \{ x \mid B_X \cap X \neq 0 \}. \tag{3.2}$$

Dilataci lze vyjádřit také jako skládání bodů dvou množin pomocí vektorového součtu (*Minkowského množinový součet*) [4]:

$$X \oplus B = \{ d \in E^2 ; d = x + b, x \in X, b \in B \}. \tag{3.3}$$

Dilataci lze vyjádřit i jako sjednocení posunutých obrazů [1]:

$$X \oplus B = \bigcup_{b \in B} X_b . \tag{3.4}$$



Obrázek 11 – jednoduchý příklad dilatace

Dilatace má několik zajímavých vlastností jako je *komutativita, asociativita, invariance vůči posunu* a řadí se mezi tzv. *rostoucí transformace(tzv. increasingness)* [1].

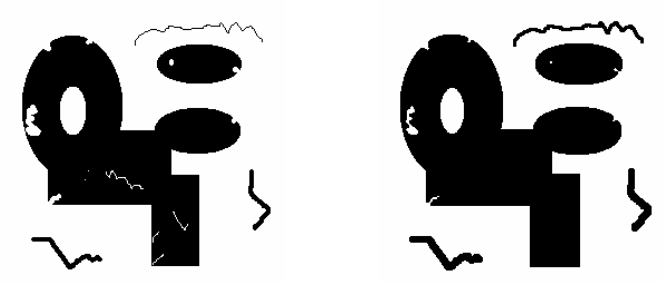
Komutativita : $X \oplus B = B \oplus X$. (3.5)

Asociativita: $X \oplus (B \oplus D) = (X \oplus B) \oplus D.$ (3.6)

Invariance vůči posunu: $X_h \oplus B = (X \oplus B)_h .$ (3.7)

Rostoucí transformace: Pokud $X \subseteq Y$, potom $X \oplus B \subseteq Y \oplus B$. (3.8)

Používá se samostatně k zaplnění malých děr, úzkých zálivů a jako základní element složitějších morfologických operací. Dilatace zvětšuje objekty na úkor pozadí. Nejčastěji se používá strukturní elementem 3x3, obsahující všech 9 bodů 8-okolí. Objekty se rozrostou o jednu slupku a díry a zálivy tloušťky 2 body se zaplní.



Obrázek 12 - původní obraz a obraz dilatovaný čtvercovým elementem 3x3

Binární eroze

Duální transformací k dilataci je *eroze*. Platí pro ně tento vztah [4]:

$$(X \ominus A)^C = X^C \oplus \check{A}\,. \tag{3.9}$$

Erozi lze formulovat otázkou (odpověď pro každý *bod x = (i,k)*) [9]:

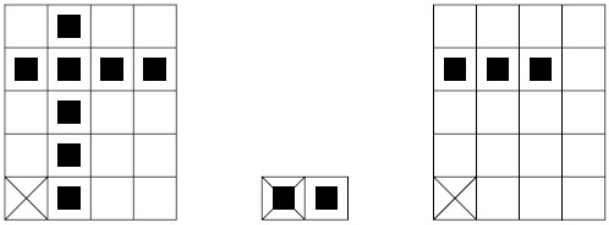
Souhlasí strukturní element B o počátku v x s objektem v f?

Mějme bodovou množinu *X* a strukturní element *B*, pak je eroze ε_B množiny *X* elementem *B* definována jako lokus bodů *x* takových, že *B* je zahrnuto v *X* pokud je počátek umístěn v *x* [4]:

$$\varepsilon_B(X) = \{x \mid B_X \subseteq X\}. \tag{3.10}$$

Alternativně lze erozi vyjádřit jako složení dvou bodových množin s využitím rozdílu vektorů (*Minkowského množinový rozdíl*) [4]:

$$X \ominus B = \{d \in E^2; d + b \in X, \forall b \in B\}\,. \tag{3.11}$$



Obrázek 13 – jednoduchý příklad eroze

Eroze je stejně jako dilatace *invariantní vůči posunu*, je *rostoucí transformací*, avšak není *komutativní*. Dále je eroze *antiextenzivní transformací* a *zachovává inkluzi* [1].

Antiextenzivní : Je-li $(0,0) \in B$, potom $X \ominus B \subseteq X$. (3.12)

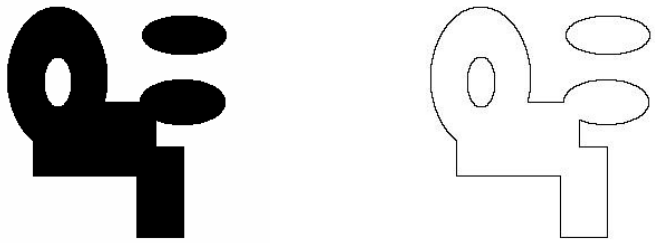
Zachovává inkluzi: Je-li $X \subseteq Y$, potom $X \ominus B \subseteq Y \ominus B$. (3.13)

Nejčastěji používaným je opět čtvercový strukturní element o velikosti 3x3 pixely. Při jeho použití zmizí objekty (čáry) tloušťky 2 a osamělé body, objekty se zmenší o 1 slupku.

Obrázek 14 - původní obraz a obraz erodovaný čtvercovým elementem 3x3

S použitím eroze lze například najít obrys objektu, který dostaneme odečtením erodovaného obrázku od původního [1]:

$$\partial X = X \setminus X \ominus B. \tag{3.14}$$



Obrázek 15 - původní obraz a obrys obrazu získaný erozí čtvercovým elementem 3x3

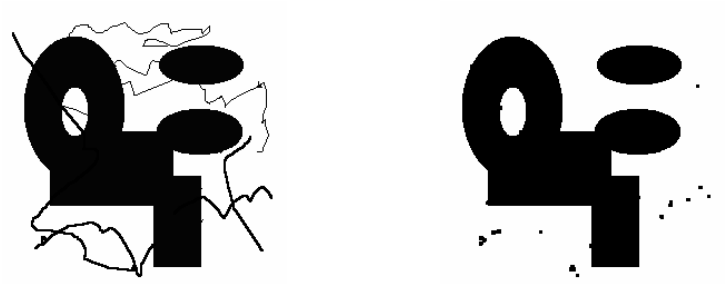
Binární otevření a uzavření

Dilatace a eroze jsou navzájem inverzní zobrazení. Jejich kombinacemi získáme další důležité morfologické transformace – *otevření* a *uzavření*. Výsledkem obou transformací je zjednodušený obraz s méně detaily. Obě tyto operace se používají pro morfologické filtrování obrazu.

Eroze aplikovaná na obraz neodstraní pouze nežádoucí struktury, ale ztenčí i všechny ostatní. Tento fakt vedl ke vzniku morfologického operátoru *otevření*, který má následným použitím dilatace navrátit původní podobu obrazu. Otevření je tedy eroze následovaná dilatací [7]:

$$X \circ B = (X \ominus B) \oplus B. \tag{3.15}$$

Pokud se obraz nezmění po otevření strukturním elementem *B*, říkáme, že je *otevřený vzhledem k B.* Otevření oddělí objekty spojené úzkou šíjí a odstraní malé detaily.

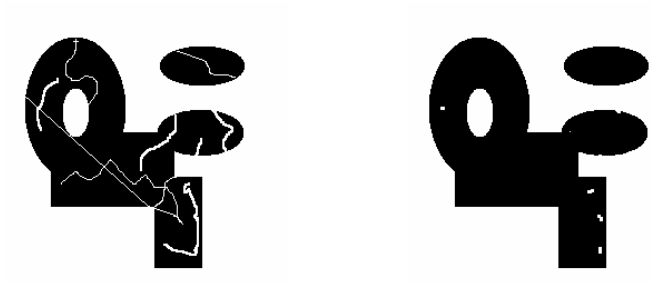


Obrázek 16 - původní obraz a obraz po operaci otevření čtvercovým elementem 3x3

Duálním operátorem k morfologickému otevření je morfologické *uzavření*, což je dilatace následovaná erozí [7]:

$$X \bullet B = (X \oplus B) \ominus B. \tag{3.16}$$

Pokud se obraz nezmění po uzavření strukturním elementem *B*, říkáme, že je *uzavřený vzhledem k B.* Uzavření spojí objekty, které jsou blízko u sebe, zaplní malé díry a úzké zálivy.



Obrázek 17 - původní obraz a obraz po operaci uzavření čtvercovým elementem 3x3

Otevření a uzavření jsou duální morfologické operace a jsou rostoucími (*increasing*) transformacemi. Důležitou vlastností těchto operací je *idempotentnost (idempotence)*, což znamená, že opakované použití těchto operací nemění předchozí výsledek [1]:

$$X \circ B = (X \circ B) \circ B, \tag{3.17}$$

$$X \bullet B = (X \bullet B) \bullet B. \tag{3.18}$$

Otevření je antiextenzivní (nějaké pixely jsou odstraněny) a uzavření je extenzivní (nějaké pixely přibudou) operací [4]. Více o idempotenci a vlastnostech morfologických filtrů je v kapitole 4.1.

3.3 Šedotónová morfologie

Šedotónová morfologie je zobecněním binární morfologie na obrázky s více bity na pixel, kde jsou místo *AND* a *OR* operace použity operace *Max* a *Min* [1]. První dvě souřadnice množiny tvoří definiční obor a třetí souřadnice odpovídá funkční hodnotě. Strukturní element je funkcí dvou proměnných a ovlivňuje, jakým způsobem se berou v úvahu hodnoty obrazu v okolí. Hodnota strukturního elementu je přičtena (nebo odečtena), když se v okolí počítá maximum (nebo minimum). V šedotónové morfologii je obraz přirovnáván k jakési mapě, kde hodnota jasu představuje výšku. Morfologické operace pak lze vyjádřit pomocí tzv. *vršků* a *stínů*.

Vršek a stín množiny

Uvažujme bodovou množinu *A* v *n*-rozměrném prostoru. Pak prvních (*n*-1) souřadnic množiny tvoří definiční obor a *n*-tá souřadnice odpovídá funkční hodnotě funkce nebo funkcí v bodě (pro šedotónové obrazy *n*=3). *Vršek množiny (top of the surface) A* je funkce definovaná na (*n*-1)-rozměrném definičním oboru. Pro každou (*n*-1)-tici je vršek hodnota zbylé poslední souřadnice množiny *A*.

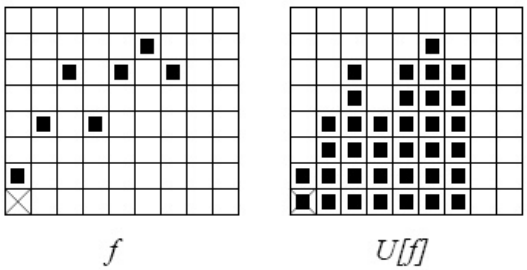
Nechť $A \subseteq \varepsilon^n$ a nechť definiční obor $F = \{ x \in \varepsilon^{n-1} \text{ pro některá } y \in \varepsilon, (x, y) \in A \}$. *Vršek* množiny *A*, označovaný *T[A]*, je zobrazením $F \to \varepsilon$ definovaným jako [1]:

$$T[A](x) = \max\{y, (x, y) \in A\}. \tag{3.19}$$

Dalším pojmem, který je třeba zavést, je *stín (umbra)* funkce *f* definovaný na (*n*-1)-dimenzionální podmnožině množiny *A*. Stínem funkce *f* je množina sestávající z vršku *f* a celého prostoru pod ním.

Nechť $F \subseteq \varepsilon^{n-1}$ a $f : F \to \varepsilon$. Stín funkce se označuje $U[f], U[f] \subseteq F \times \varepsilon$ a je definovaný vztahem [1]:

$$U[f] = \{(x, y) \in F \times E, y \leq f(x)\}. \tag{3.20}$$



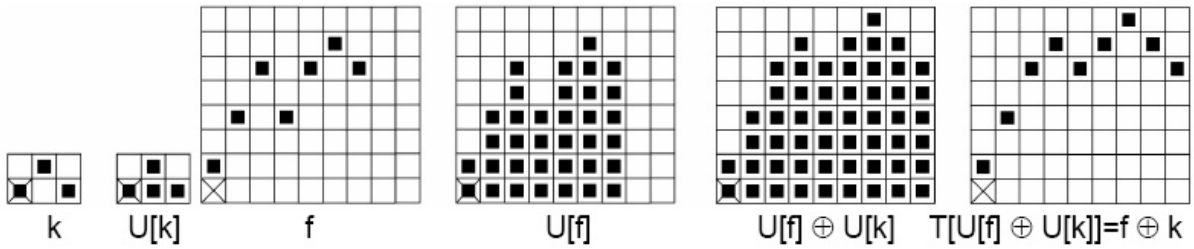
Obrázek 18 – 1D funkce a její stín

Stínem stínu funkce *f* je opět stín.

Šedotónová dilatace

Šedotónovou dilataci lze definovat jako vršek dilatace jejich stínů. Nechť $F, K \subseteq \varepsilon^{n-1}$, $f : F \to \varepsilon$ a $k : K \to \varepsilon$. Šedotónová dilatace $\oplus$ funkce *f* s funkcí *k*, $f \oplus k$: $F \oplus K \to \varepsilon$ je definována jako [1]:

$$f \oplus k = T\{U[f] \oplus U[k]\}. \tag{3.21}$$



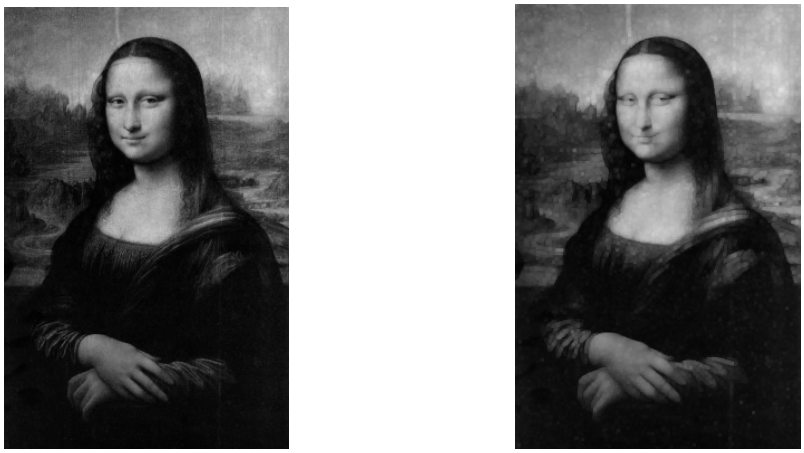
Obrázek 19 – 1D příklad: strukturní element, stín strukturního elementu, funkce, stín funkce, …

Efektivněji pro implementaci lze dilataci vyjádřit jako maximum hodnot pixelů zasažených strukturním elementem.

$$(f \oplus k)(x) = \max\{f(x-z) + k(z), z \in K, x - z \in F\} \tag{3.22}$$

Pokud budeme předpokládat strukturní element *B*, který zahrnuje konečný počet pixelů a je konvexní a ohraničený, pak lze operaci zapsat takto [4]:

$$\mathrm{D}_{\mathrm{G}}(A, B) = \max_{[j,k] \in B}\{a[m-j, n-k]\} = \max_{B}(A). \tag{3.23}$$



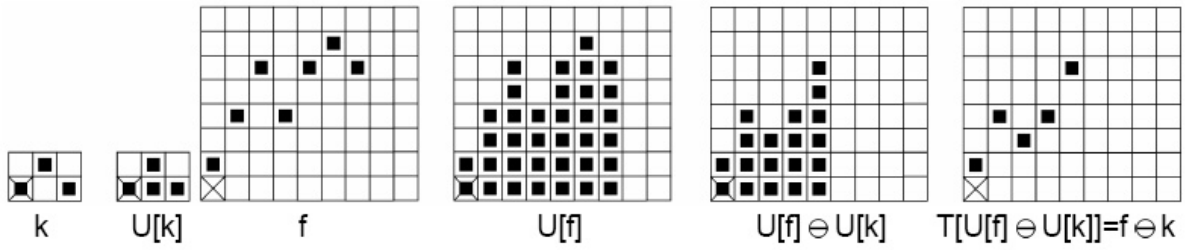
Obrázek 20 - původní obraz a obraz dilatovaný čtvercovým elementem 3x3

Pro operaci šedotónové dilatace lze použít také nerovinné strukturní elementy, které mají šedotónové hodnoty pro každý bod svého definičního oboru. Tyto strukturní elementy se však kvůli značné výpočetní náročnosti v aplikacích používají jen málo.

Šedotónová eroze

Šedotónová eroze dvou funkcí nejprve najde jejich stíny, ty pak eroduje binární erozí a nakonec vypočte vršek množiny jako výsledek. Nechť $F, K \subseteq \varepsilon^{n-1}$, $f : F \to \varepsilon$ a $k : K \to \varepsilon$. Šedotónová eroze $\ominus$ funkce *f* s funkcí *k*, $f \ominus k$: $F \ominus K \to \varepsilon$ je definována jako [1]:

$$f \ominus k = T\{U[f] \ominus U[k]\}. \tag{3.24}$$



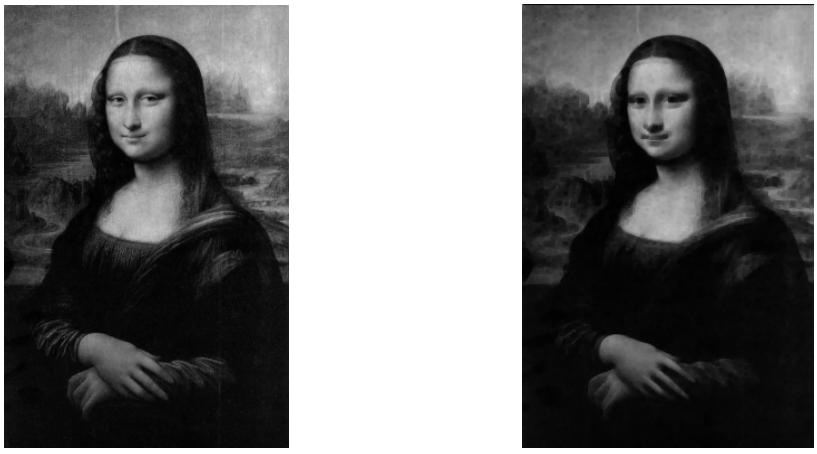
Obrázek 21 – 1D příklad : strukturní element, stín strukturního elementu, funkce, stín funkce, …

Aby se snížila výpočetní náročnost, probíhá skutečný výpočet jinak, jako hodnota pixelů zasažených strukturním elementem [4]:

$$(f \ominus k)(x) = \min\{f(x+z) - k(z), z \in K, x+z \in F\}. \tag{3.25}$$

Pro zjednodušení budeme opět předpokládat strukturní element *B,* který zahrnuje konečný počet pixelů a je konvexní a ohraničený:

$$\mathrm{E}_{\mathrm{G}}(A,B) = \min_{[j,k] \in B}\{a[m-j, n-k]\} = \min_{B}(A). \tag{3.26}$$



Obrázek 22 - původní obraz a obraz erodovaný čtvercovým elementem 3x3

Stejně jako pro šedotónovou dilataci lze i pro erozi použít také nerovinné strukturní elementy. Může však dojít k tomu, že některé vypočtené hodnoty budou záporné. Opět se ze stejných důvodů nepoužívají příliš často [4].

Šedotónové otevření a uzavření

Šedotónové otevření a uzavření lze zavést stejným způsobem jako v binární morfologii. Šedotónové otevření je definováno takto [1]:

$$f \circ k = (f \ominus k) \oplus k. \tag{3.27}$$

Ze zjednodušené definice lze šedotónové otevření popsat tímto vztahem:

$$O_G(A,B) = \max_B(\min_B(A)). \tag{3.28}$$

Podobně pro šedotónové uzavření platí [1]:

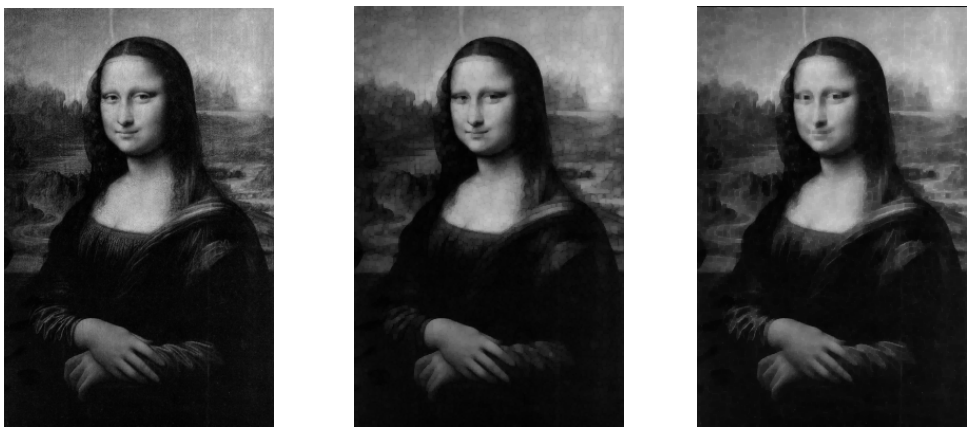
$$f \bullet k = (f \oplus k) \ominus k. \tag{3.29}$$

Případně jej lze popsat takto:

$$C_G(A,B) = \min_B(\max_B(A)). \tag{3.30}$$

Dualita mezi otevřením a uzavřením je vyjádřena vztahem [1]:

$$-(f \circ k)(x) = [(-f) \bullet \breve{k}](x). \tag{3.31}$$



Obrázek 23 – původní obraz, obraz po operaci otevření a uzavření čtvercovým elementem 3x3

Šedotónové otevření obrazu *f* strukturním elementem *k* si lze představit jako posouvání funkce *k* krajinou *f* [1]. Poloha všech nejvyšších bodů nějakou částí *k* při posunu dává otevření. Podobná interpretace existuje i pro uzavření. Šedotónové otevření a uzavření se používá k extrakci částí obrazu s daným tvarem a šedotónovou strukturou.

Homeomorfismus stínu, vlastnosti dilatace a eroze

Operace vršek množiny je vždy levou inverzí operace stín [1]. Tyto operace poskytují intuitivní vztah mezi šedotónovou a binární morfologií. Věta o homeomorfismu stínů říká, že stín je homeomorfismem ze šedotónové do binární morfologie. Nechť $F, K \subseteq \varepsilon^{n-1}$, $f : F \to \varepsilon$ a $k : K \to \varepsilon$. Potom platí [1]:

$$U[f \oplus k] = U[f] \oplus U[k], \tag{3.32}$$

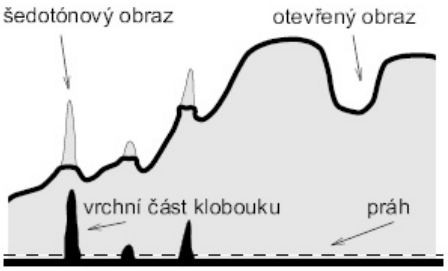
$$U[f \ominus k] = U[f] \ominus U[k]. \tag{3.33}$$

Homeomorfismus stínů lze použít pro odvození vlastností operací šedotónové morfologie. Šedotónové operace se vyjadřují pomocí vršků a stínů a převedou se díky homeomorfismu stínů na operace nad binárními obrazy. Díky existenci tohoto zobrazení lze říci, že šedotónové operace mají stejné vlastnosti jako operace binární.

Transformace vrchní části klobouku

Množinový rozdíl mezi otevřením šedotónového obrazu *X* strukturním elementem *K* a původním obrazem *X* je novou operací zvanou *vrchní část klobouku* (*top hat transformation*) [1]. Tato operace se používá jako jednoduchý nástroj pro segmentaci objektů, které se v šedotónovém obraze liší od pozadí, když se jas pozadí pomalu mění. Části obrazu, které se nevejdou do strukturního elementu *K* použitého pro otevření, se odstraní. Po odečtení otevřeného obrazu od původního se objeví jen odstraněné části obrazu, ty lze pak nalézt prahováním. Transformace najde pouze vrchní část „klobouku“, když je strukturní element větší než otvor pro hlavu (*light top hat transformation*) [4].

$$LightTopHat(X, K) = X - (X \circ K) = X - \max_K\left(\min_K(X)\right) \tag{3.34}$$



Obrázek 24 – transformace vrchní části klobouku

Existuje i modifikace této operace, která vyhledává naopak tmavé části obrazu na proměnlivém světlém pozadí (*dark top hat transformation*). Ta se provede odečtením původního obrazu od uzavřeného.

$$DarkTopHat(X, K) = (X \bullet K) - X = \min_K\left(\max_K(X)\right) - X \tag{3.35}$$

Morfologický gradient

Poslední často používanou morfologickou operací nad šedotónovými obrazy je *morfologický gradient*. Ten detekuje prostorově náhlé změny v intensitách pixelů.

$$Gradient(X, K) = \max_K(X) - \min_K(X) \tag{3.36}$$

3.4 Morfologie nad barevným obrazem

Na rozdíl od šedotónových obrázků, kde zpracovávané hodnoty představují skalár, u barevných obrázků je jeden pixel reprezentován vektorem (o třech složkách). V praxi se většinou pracuje s RGB obrázky nebo s obrázky v modelu HSV (HSB). Mezi dvěma vektory (o více než jedné dimenzi) nelze určit menší nebo větší z nich.

Barevná morfologie se dá podle způsobu řazení (podle Barnetta) klasifikovat do 4 skupin [10]: *marginální uspořádání (marginal ordering), redukované uspořádání (reduced ordering), částečné uspořádání (partial ordering) a podmíněné uspořádání (conditional ordering).*

Marginální uspořádání (marginal ordering)

Marginální uspořádání částečně řeší problém uspořádání. Každý kanál obrazu je samostatně zpracován obdobným způsobem jako šedotónový obraz:

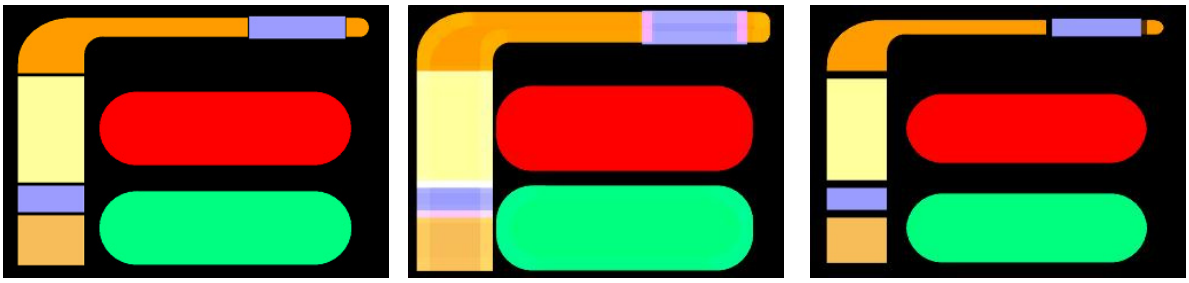
$$\varepsilon_B(f) = [\varepsilon_{B1}(f_1), \varepsilon_{B2}(f_2), \ldots, \varepsilon_{Bm}(f_m)] \text{ (eroze)}, \tag{3.37}$$

$$\delta_B(f) = [\delta_{B1}(f_1), \delta_{B2}(f_2), \ldots, \delta_{Bm}(f_m)] \text{ (dilatace)}, \tag{3.38}$$

kde $B=(B_1,B_2,\ldots B_m)$ reprezentuje strukturní element. Operace, které se používají u tohoto uspořádání se nazývají *component-wise operators.* Toto zpracování může výrazně změnit spektrální kompozici vstupního obrazu. Například je možné, že objekt zmizí v *R* a *G* kanálu a zůstane pouze v *B* kanálu. Efekty vytvářené tímto přístupem jsou pro mnoho aplikací nepřípustné, ale i přesto je marginální uspořádání ve zpracování obrazu používáno a to nejen pro RGB obrazy.

Obrázek 25 - původní obraz, obraz dilatovaný a obraz erodovaný čtvercovým elementem 3x3 (RGB)

Aby se předešlo výše zmíněným problémům, je možné reprezentovat všechny vektory dvěma barvami, které by představovaly objekt a jeho pozadí [4]. Potom by se s takovým obrazem pracovalo stejně jako s binárním.



Obrázek 26 - původní obraz, obraz dilatovaný a obraz erodovaný čtvercovým elementem 3x3 (RGB)

Částečné uspořádání (partial ordering)

V tomto řadícím schématu jsou barevné vektory seskupeny do různých podmnožin a to takových, že členové všech vzniklých skupin mají stejné „řadící“ hodnoty (hodnoty, na které byly namapovány) [10]. Sem bychom mohli zařadit *řazení podle vzdálenosti (ordering by distance)*, kde se pracuje s pozicí vektoru v prostoru.

Obrázek 27- dilatace a eroze použitím částečného uspořádání komponentní x vektorový základ (HSV)

Podmíněné uspořádání (conditional ordering)

Jedno ze schémat pro podmíněné uspořádání je tzv. lexikografické uspořádání [10]. To se používá v mnoha studiích pro řazení v barevných obrazech. V tomto schématu, je vybrána nejvýznamnější složka a řazení se provádí podle této hodnoty. Takové komponenty, které mají stejnou hodnotu, pak mají i stejnou úroveň. Poté jsou porovnávány hodnoty druhých nejvýznamnějších složek. Dále bychom sem mohli zařadit i *kanonické řazení (canonical ordering)*, kde všechny tři složky barevného prostoru musí mít vyšší nebo nižší hodnoty než hodnoty ostatních vektorů.

Redukované uspořádání (reduced ordering)

Aby bylo možné přistupovat u obrázku reprezentovaném vektory jinak než jako k „binárnímu“, je nutné, aby nad vektory byla zavedena relace uspořádání [10]. Dalším způsobem zpracování, který se někdy používá je redukované uspořádání. Jeho název je odvozen od skutečnosti, že hodnoty vektoru každého pixelu jsou redukovány do jedné skalární hodnoty. Takové hodnoty pak mohou být řazeny.

Nechť $x_1,x_2,\ldots,x_n$ je množina multivarietních vzorků, kde vektor x_i je vektor v R^p. N vzorků seřadíme podle schématu redukovaného uspořádání. Prvním krokem je namapovat každé x_i na skalární hodnotu $d_i=d(x_i)$, kde $d: R^p \rightarrow R$. Poté co získáme d_i pro každé i, lze seřadit vektory $x_1,x_2\ldots,x_n$ na základě $d_1..d_n$ následovně:

$$x_{(1)} \leq x_{(2)} \leq x_{(3)} \leq \ldots \leq x_{(n)}, \tag{3.39}$$

kde $x_{(r)}$ je hodnota korespondující se skalární hodnotou $d_{(r)}$, a $d_{(r)}$ je r-tý nejmenší prvek množiny $\{ d_1..d_n \}$.

Podle výše uvedeného lze definovat vektorové morfologické operace pro barevný obrázek takto:

- Strukturní element je zde definován jako množina H
- Funkce pro redukované uspořádání $d: R^3 \rightarrow R$

Operace vektorové dilatace se reprezentuje symbolem $\oplus_v$. Hodnota vektorové dilatace f elementem H v bodě (x,y) je definováno jako [10]:

$$(f \otimes_v H)(x,y) = a, \tag{3.40}$$

kde

$$a \in \{ f(r,s) : (r,s) \in H_{(x,y)} \} \tag{3.41}$$

a

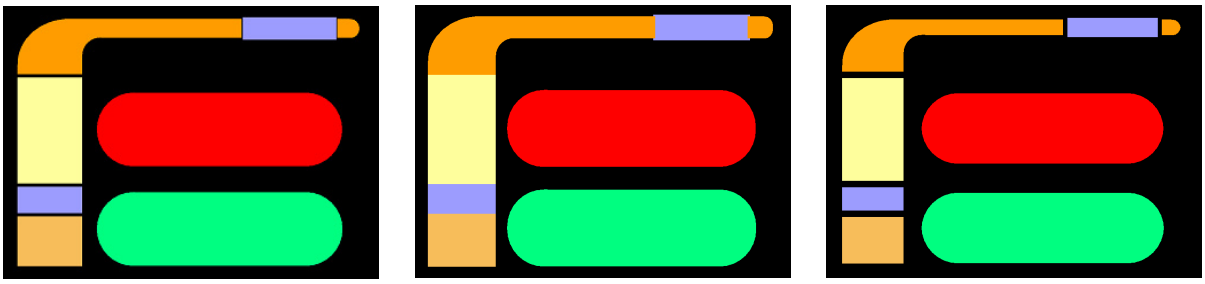
$$d(a) \geq d(f(r,s)) \forall (r,s) \in H_{(x,y)}. \tag{3.42}$$

Podobně je reprezentována vektorová eroze, která se označuje symbolem $\ominus_v$. Otevření a uzavření je definováno stejně jako u šedotónových obrazů [10].

Důležité je uvědomit si, že výstupní obraz nezávisí pouze na vstupním obrazu a na strukturním elementu H. Je nutné mít ještě navíc definovanou funkci f pro redukované uspořádání. Většinou je nejlepší využít znalosti lidského vnímání jako je jas a použít jej jako metriku pro funkci redukovaného uspořádání. Například můžeme použít lineární kombinaci tří hodnot. Pak můžeme napsat [10]:

$$d(f(x,y)) = a_R f_R(x,y) + a_G f_G(x,y) + a_B f_B(x,y), \tag{3.43}$$

pokud je obrázek filtrován v RGB barevném prostoru. a_R=0,299, a_G=0,587 a a_B=0,114 a *d(f(x,y))* je jasový obraz [11]. Hodnoty a_R, a_G a a_B mohou být vybrány ke zvýraznění nebo k potlačení některé barvy. Podobně, když a_R=1, a_G=0 a a_B=0, potom je efekt *multiscale otevření* takový, že budou potlačeny objekty s výrazně červeným obsahem.



Obrázek 28 - původní obraz, obraz dilatovaný a obraz erodovaný čtvercovým elementem 9x9(YCbCr-Y)

Dále je možné se setkat řazením podle jedné složky barevných modelů *(ordering by one component)* a to třeba podle jasové složky modelů HSV, HSL nebo YCbCr.

Obrázek 29 - původní obraz, obraz dilatovaný a obraz erodovaný čtvercovým elementem 3x3(YCbCr-Y)

3.5 Transformace tref či miň a skelet

Transformace *tref či miň* byla první definovanou morfologickou operací. Na rozdíl od již popsaných morfologických operací zahrnuje strukturní elementy dvou množin [1]. První množina má studovaný objekt trefit zatímco druhá minout. Operace tref či miň způsobuje tzv. ztenčování objektů. Sekvenčně ztenčovat lze až na skelet (kostra objektu).

Transformace tref či miň

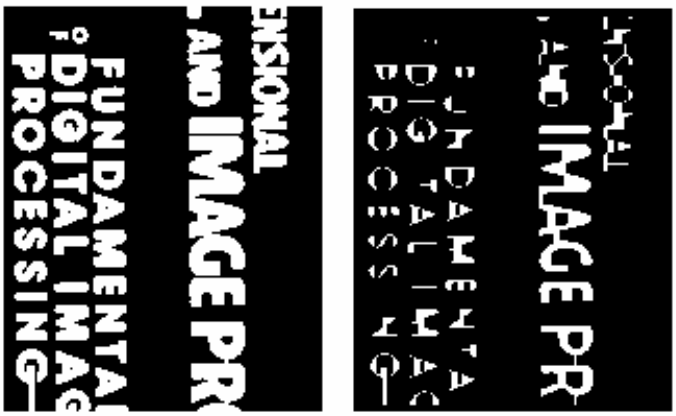
Transformac*e tref či miň* je morfologický operátor indukující shodu strukturního elementu a části obrazu. Strukturní element představuje vzor, který se vyhledává, a může například sloužit k vyhledávání rohů, hranic objektů nebo pro ztenčování. Binární operace dilatace a eroze využívaly strukturní element *B*, kterým byl testován výskyt bodů v obraze *X*. Nyní bude potřeba testovat i to, zda některé body do *X* nepatří. K tomu slouží složený strukturní element, což je dvojice disjunktních množin $B = (B_1;B_2)$. Transformace tref či miň označovaná $\otimes$ je definována jako [1]:

$$X \otimes B = \{x : B_1 \subset X, B_2 \subset X^C\}. \tag{3.44}$$

Má-li být bod *x* ve výsledné množině, musí být současně splněny dvě podmínky. Musí být část B_1 složeného strukturního elementu s reprezentativním bodem v poloze x obsažena v *X*. A dále nesmí být část B_2 složeného strukturního elementu obsažena v X^C. Transformaci tref či miň lze vyjádřit i pomoci erozí a dilatací:

$$X \otimes B = (X \ominus B_1) \cap (X^C \ominus B_2) = (X \ominus B_1) \setminus (X \oplus \breve{B}_2). \tag{3.45}$$

Používá se pro ztenčování, ztlušťování nebo hledání kostry objektu. Existuje také rozšíření této operace pro šedotónovou morfologii [24].

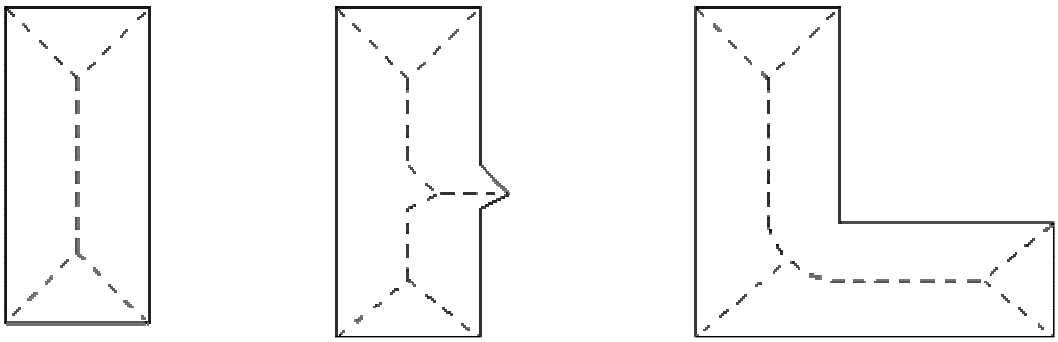


Obrázek 30 - původní obraz, obraz po operaci tref či miň čtvercovým elementem 3x3

Můžeme se setkat také s podobnou operací, která je jakýmsi zobecněním morfologických operací, jelikož pomocí ní mohou být všechny operace realizovány. Její anglický název je *hit and miss transform*.

Skelet

Velice důležitým krokem v reprezentaci strukturálního tvaru planární oblasti je redukovat ji do grafu. Takto vzniklý graf nazýváme *skelet* (kostra) daného objektu. Neformálně lze skelet definovat jako čárovou reprezentaci objektu, která má tloušťku 1 pixel, prochází středem objektu a zachovává jeho topologii. To ovšem není vždy realizovatelné.



Obrázek 31 – skelety tří různých objektů

Můžeme se setkat s různými definicemi skeletů [4]:

Eukleidovské skelety:

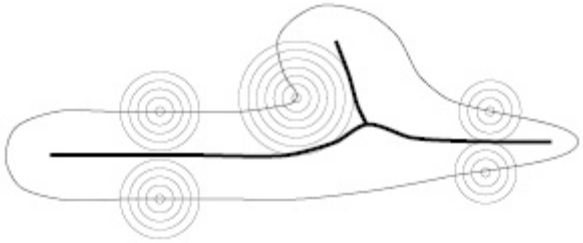
- Požár trávníku a jeho šíření.
- Vzdálenostní funkce.
- Maximální kruhy.
- Minimální cesty.
- Otevření.

Diskrétní skelety:

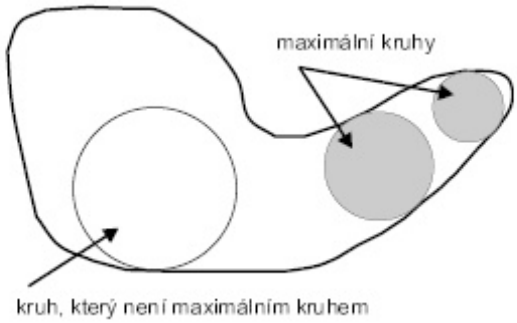
- Otevření.
- Homotopické sekvenční ztenčování.
- Homotopické ztenčování nezávislé na pořadí.
- Vzdálenostní funkce.
- Skeleton pruning.
- Skelet pomocí zón vlivu.

S první myšlenkou přišel ve zpracování obrazu Blum, tehdy pod názvem *střední osa* (*medial axis transform*) [1]. Znázornil ji představou šíření ohně na suchém trávníku. Řekněme, že oblastí, jejíž

skelet chceme získat, je suchý trávník a zapálíme jej v jednom okamžiku po celém obvodu. Předpokládejme, že se oheň se šíří konstantní rychlostí. Skelet budou tvořit všechny body, kde se setkají dva a více ohňů. Formální definice skeletu se opírá o pojem *maximálního kruhu*. Maximální kruh *B(p; r)* se středem *p* a poloměrem *r*, $r \geq 0$, je množina bodů, jejichž vzdálenost *d* od středu je menší nebo rovna *r*. Kruh *B* vepsaný do množiny *X* se nazývá *maximálním kruhem*, právě když se hranice množiny dotýká ve dvou a více bodech, což znamená, že pro dané místo dotyku již kruh nelze zvětšit. Pro diskrétní obrazy závisí definice vzdálenosti na použité mřížce a definici souvislosti.



Obrázek 32 - Skelet jako místa na suchém trávníku, kde se setkají dva a více šířících se ohňů



Obrázek 33 - vepsaný kruh a maximální vepsaný kruh v euklidovské rovině

Některé praktické způsoby nalezení skeletu jsou např. tyto:

- Sekvenční ztenčování strukturním elementem L Golayovy abecedy.
- Sekvenční ztenčování strukturním elementem E Golayovy abecedy.
- Vincentův algoritmus.
- Vzdálenostní funkce.
- Maximální kruhy.
- A mnoho dalších…

4 Některá použití morfologické analýzy obrazu

Počítačové zpracování a analýza obrazu mají své počátky na konci 60. let minulého století. Prvními technikami používanými pro zpracování těchto dat byly především operátory pro 2-dimenzionální data [4]. Od té doby se rozšířila oblast použití obrazové analýzy téměř do všech technických a vědeckých oblastí, ve kterých lze narazit na mnoho různých specifických problémů a způsobů jejich řešení. Pro analýzu obrazu se používají i morfologické operace, které mají uplatnění např. ve filtrování a segmentaci obrazu, při analýze textur nebo pro různé geodetické transformace.

4.1 Filtrování obrazu

Morfologické filtry jsou nelineární a lze je definovat jako operátory založené na lokální transformaci. Jsou vhodné ke dvěma druhům filtrování [4].

Případy použití morfologických filtrů

Prvním je morfologický filtr používaný k restauraci obrazů poškozených nějakým typem šumu. Existuje mnoho lineárních filtrů k odstranění šumu, ale oproti nelineárním filtrům (např. právě morfologickým) selhávají tím, že rozmazávají ostré hrany. Zašuměný obraz se předzpracuje morfologickým filtrem například před následnou detekcí hran nebo segmentací.

Obrázek 34 - obraz zašuměný šumem pepř a sůl, obraz filtrovaný morfologickým filtrem otevření-zavření čtvercem 2x2 a obraz filtrovány mediánovým filtrem

Dalším je morfologický filtr, který umožňuje selektivně odstranit objekty nebo struktury určitých vlastností. Ostatní obsah obrazu zůstává zachován. Výběr je založen na geometrii a „lokálním kontrastu“ objektů v obraze (například světlý objekt na tmavém pozadí). V tomto smyslu lze považovat morfologické filtry za první krok k interpretaci obrazu.

Definice morfologických filtrů

Klíčovou vlastností morfologických filtrů je *idempotence* [4]. Aplikuje-li se na obraz morfologický filtr, který je pak na výsledek filtrování opět použit, nedojde již k žádné změně. Toto je možné přirovnat k prosívání přes síto. Morfologické filtry sdílejí s procesem prosívání i další nezbytnou vlastnost, kterou je „přírůstkovost“ (*increasingness*). Tato vlastnost zajišťuje závislost na pořadí, v jakém jsou použity na obraz různé filtry. U morfologických filtrů lze narazit i na další vlastnosti jako třeba „rozšiřování“ (*extensivity*), která nám říká, že výstupní obraz bude vždy „větší“ než obraz původní. Mezi základní morfologické filtry patří např. již dříve zmíněné nerozšiřující (antiextenzivní) otevření a rozšiřující (extenzivní) uzavření. Filtry, které nejsou ani rozšiřující ani nerozšiřující, lze členit podle jiných vlastností na *sup-filtry a inf-filtry*. Pokud lze filtr zařadit mezi sup-filtry i inf-filtry, je tento filtr označován jako silný filtr a je vždy rozšiřující. Dále je možné se setkat s pojmy jako *over-filter* a *under-filter*, přičemž tato označení říkají, jak se chovají filtry, nesplňují-li idempotenci, ale jen „přírůstkovost“ a jsou-li použity vícekrát za sebou.

Návrh základních morfologických filtrů

Nové morfologické filtry vznikají kombinováním elementárních filtrů [4]. Nejsou však povoleny všechny kombinace. Například kombinace dvou otevření nemusí být ani otevření ani filtr (nemusí být splněna idempotence). Existují 3 metody pro výstavbu nových filtrů z již existujících transformací: paralelní, sekvenční a iterativní kombinace.

Paralelní kombinací je například nezávislé provedení 2 operací na daném obraze. Nad dvěma výslednými obrazy je pak provedena operace sjednocení (*point-wise maximum*) nebo průniku (*point-wise minimum*).

Jak už bylo zmíněno, kompozicí dvou filtrů nemusí být vždy filtr. Pokud jsou však filtry uspořádané dvojice (většinou dvojice otevření a uzavření) a jsou aplikovány na obraz sekvenčně, je výsledný filtr vždy také filtr.

Některé další filtry

Dále je možné se setkat s pojmem alternující (střídající se) sekvenční filtry. Nechť γ_i je otevření a δ_i je k němu duální uzavření. Potom jsou následující 4 sekvenční kombinace morfologické filtry [4]:

$$m_i = \gamma_i \delta_i, \; r_i = \delta_i \gamma_i \delta_i, \; n_i = \delta_i \gamma_i, \; s_i = \gamma_i \delta_i \gamma_i . \tag{4.1}$$

Pak alternující sekvenční filtr velikosti i je definován jako kombinace jednoho z těchto filtrů, začínající filtrem velikosti jedna a končící filtrem velikosti i [4]:

$$M_i = m_i \ldots m_2 m_1 , \; R_i = r_i \ldots r_2 r_1 , \; N_i = n_i \ldots n_2 n_1 , \; S_i = s_i \ldots s_2 s_1 . \tag{4.2}$$

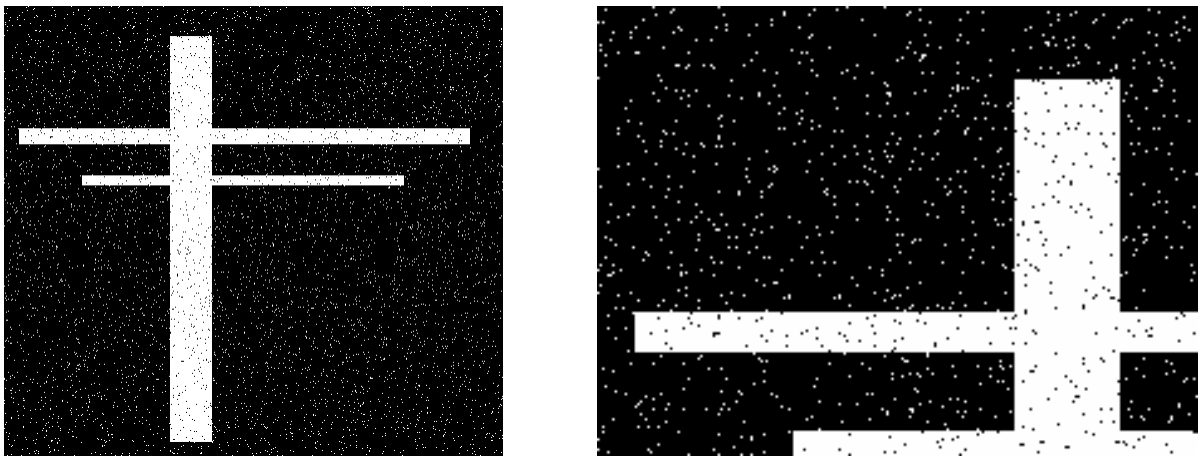
Dalšími jsou tzv. *self-dual* filtry, využívající duality morfologických operací.

4.2 Příklady filtrování obrazu - odstranění šumu

Jak už bylo zmíněno, morfologické filtry lze použít k odstranění šumu, aniž by došlo k rozmazání hran. Stejně jako ostatní filtry, ani morfologické nelze použít na všechny typy šumu. Vybrala jsem některá vzorové příklady, které demonstrují princip jejich použití. Binární a šedotónové obrázky jsou poškozeny uměle přidaným šumem typu „pepř a sůl“ (poškození náhodnými pixely s minimálním a maximálním jasem v obraze). U barevné fotografie je to skutečný šum, vzniklý v důsledku nedostatečného osvětlení a nekvalitního fotoaparátu.

Binární obrazy

Tento binární syntetický obrázek je poškozen šumem typu „pepř a sůl“ (náhodné černé a bílé pixely) [25].

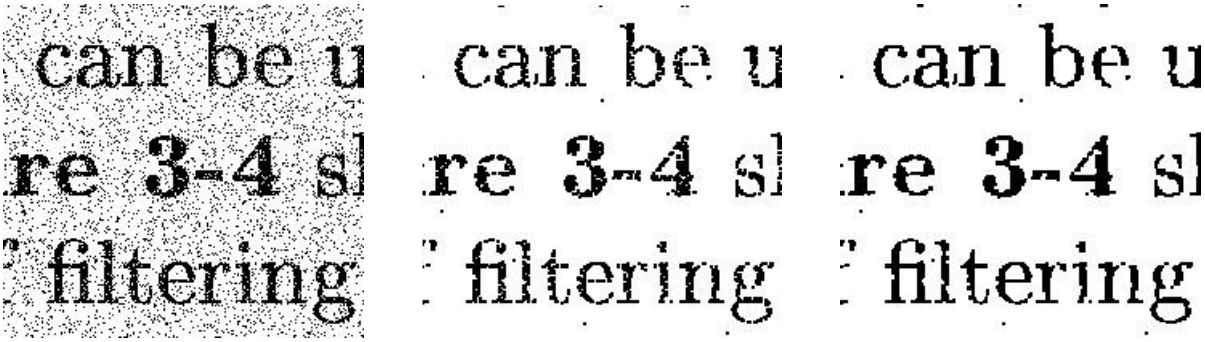


Obrázek 35 – obraz poškozený šumem“ pepř a sůl“ a jeho detail

Nejdříve byla použita operace otevření čtvercovým strukturním elementem o velikosti 3 pro odstranění šumu v bílých pruzích. Šum z pozadí byl pak odstraněn pomocí operace uzavření čtvercovým strukturním elementem o velikosti 9.

Obrázek 36 – detail obrazu po operaci otevření a obraz zcela bez šumu

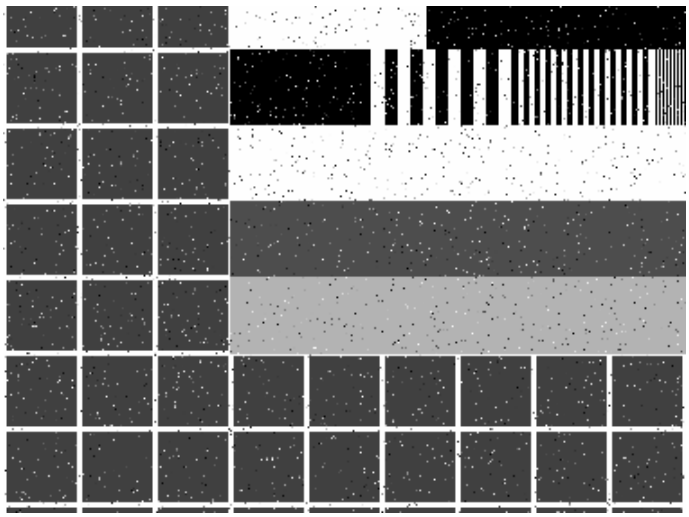
Dalším binárním obrazem je prahovaná fotografie opět s přidaným šumem „pepř a sůl“ [27]. K jeho úpravě jsem opět použila filtr otevření-zavření čtvercovým strukturním elementem o velikosti 2. Operace otevření odstranila černé nežádoucí pixely a uzavření zaplnila bílé díry.



Obrázek 37 – původní poškozený obraz, obrazu po operaci otevření a obraz zcela bez šumu

Šedotonové obrazy

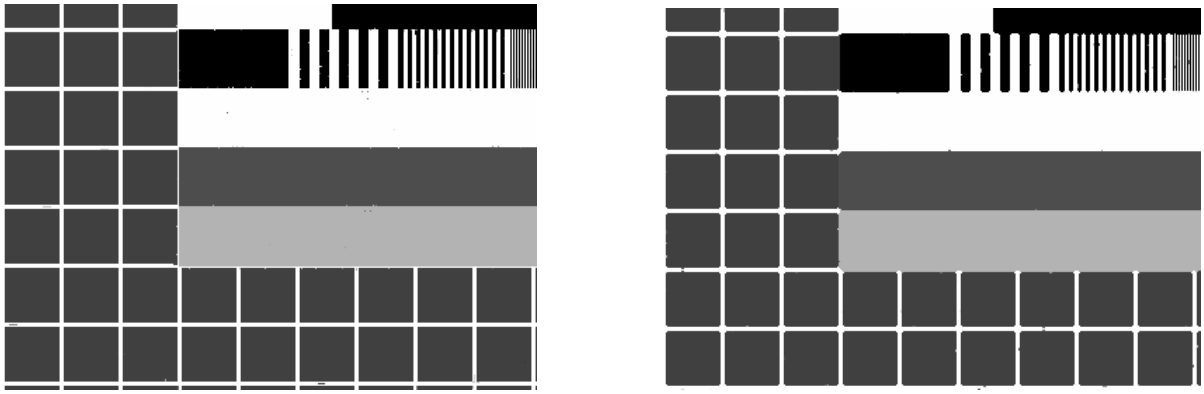
Tento syntetický šedotónový obraz byl poškozen opět šumem „pepř a sůl“ [28].



Obrázek 38 – obraz poškozený šumem“ pepř a sůl“

Pro vyčištění jsem použila dvě filtrování nezávisle na sobě a pak provedla jejich sjednocení. Nejprve jsem na zašuměný obraz aplikovala operaci otevření strukturním elementem LINE_X (viz. kapitola 5.2) o velikosti 5 (maticí o velikosti 5x1) a na výsledek pak uzavření strukturním elementem LINE_X o velikosti -5 (maticí o velikosti -5x1), aby zůstaly zachovány vodorovné čáry. Na původní obraz jsem pak použila stejné operace se stejnými velikostmi, jen se strukturním elementem LINE_Y (zachování svislých čar). Nad dvěma vzniklými obrazy jsem provedla operaci množinového sjednocení. Výsledný obraz není zcela bez šumu, ale zachovává základní geometrii prvků. Naproti tomu mediánový filtr šum odstranil, ale na úkor ztráty některých vlastností (například „zaoblené spoje“ v průsečících čar).

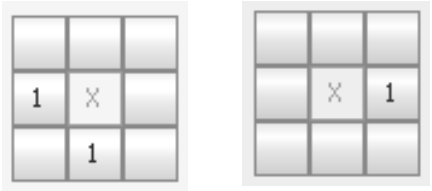
Obrázek 39 – obrazy po otevření a uzavření elementem LINE_X (vodorovné čáry) a LINE_Y (svislé čáry)



Obrázek 40 – výsledný obraz po sjednocení předchozích a obraz pro filtraci mediánovým filtrem

Obrázek 41 – zašuměná fotografie

Šedotónová fotografie je poškozena méně než pětiprocentním přidaným šumem typu „pepř a sůl“ [26]. Pro více zašuměnou fotku již docházelo při zpracování ke značnému rozmazání (poškozené



Obrázek 42 – použité strukturní elementy

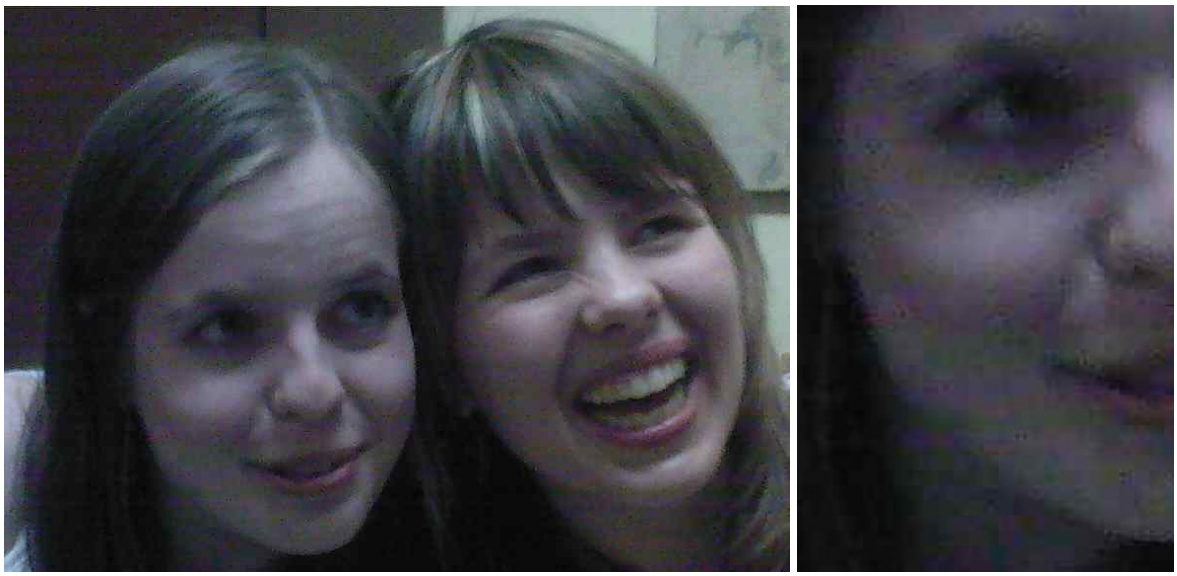
pixely spolu častěji sousedily a musel být použit větší strukturní element), proto bylo použití morfologických operací nevhodné. Na fotku jsem použila nejdříve operaci uzavření pro odstranění všech nežádoucích černých pixelů a pak otevření na zbytky bílých pixelů. Operace byly provedeny se zobrazenými strukturními elementy, které byly experimentálně zvoleny tak, aby se odstranil všechen šum, ale obrázek byl co nejméně rozmazaný.

Obrázek 42 – fotografie po provedení operace uzavření a finální fotografie bez šumu

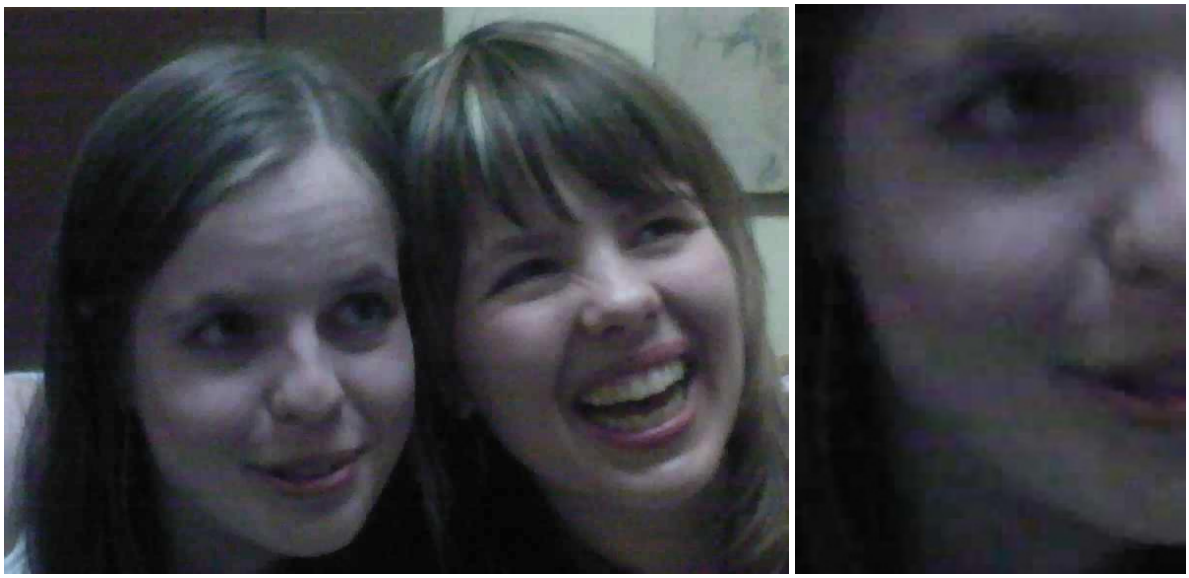
Barevné obrazy

Barevná fotografie (toto je jen výřez) byla pořízena za slabšího osvětlení mobilním telefonem, proto je trošku zašuměná a tmavá. Při úpravě jsem použila filtr uzavření-otevření čtvercovým strukturním elementem o velikosti 3. Nejdříve uzavření fotku trošku zesvětlilo a odstranilo černý šum. Zbylý bílý šum byl odstraněn operací otevření. Obě operace byly provedeny marginálním přístupem v modelu RGB.

Tento filtr je možné použít na odstranění výstřelového šumu v obraze s velkým rozlišením, aby nedošlo k viditelnému rozmazání strukturním elementem.



Obrázek 43 – původní fotografie a její detail



Obrázek 44 – upravená fotografie a její detail

Kdy použít morfologické filtry

Pokud se jedná o syntetické obrázky s nějakými pravidelnými tvary, lze při odstraňování šumu využít výše popsaných postupů, které zachovávají geometrii objektů i tam, kde by jiné filtry rozmazaly hrany. Někdy je však možné, že vhodná kombinace operací a strukturních elementů pro úplné odstranění šumu neexistuje.

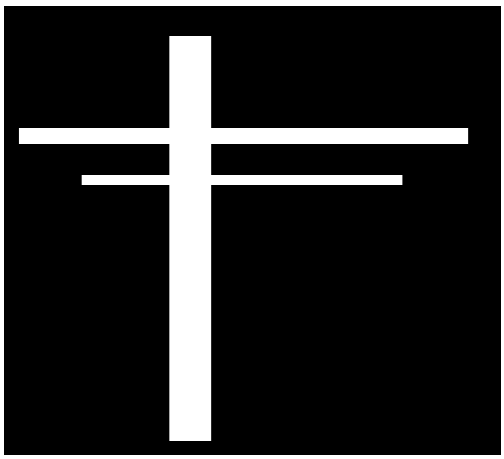
U fotografií a mnohotvárnějších obrázků záleží na „velikosti“ šumu vůči velikosti (rozlišení) obrázku a objektů v něm. Pokud je šum tvořen shluky více pixelů nebo je jeho intenzita větší, je nutné použít pro jeho odstranění větší strukturní element, který může některé objekty rozmazat nebo je zcela odstranit. Je tedy vhodné použít co nejmenší strukturní element (nejlépe izotopický), aby došlo k minimálnímu rozmazání, a zároveň byl šum odstraněn. Většinou neodstraníme šum všechen, případně se musíme spokojit s rozmazáním obrazu.

Morfologické filtry kromě odstranění šumu také umožňují zesvětlení světlých (dilatace) nebo ztmavení tmavých částí (eroze) obrazu. Samotné operace dilatace a eroze se často nepoužívají, po jejich aplikaci je výsledek extrémní – příliš světlý nebo příliš tmavý. Ve většině případů se používají filtry, které jsou kombinacemi operací uzavření a otevření. Uzavření je extenzivní operace, a proto obraz mimo jiné mírně zesvětluje. Naopak otevření je operací antiextenzivní, a tak obraz trošku ztmaví. U barevných obrazů je toto zesvětlení a ztmavení relativní, zde záleží na použitém barevném model a vlastnostech jeho složek.

4.3 Příklady filtrování obrazu – selektivní odstranění objektu

Kromě odstranění šumu v obraze lze morfologický filtr použít k selektivnímu odstranění objektů nebo struktury určitých vlastností. Ostatní obsah obrazu zůstává zachován.

Binární obrazy



Úkolem je rozdělit tento kříž na vodorovné a svislý pruh [25]. To je úloha právě pro selektivní odstranění buď svislého nebo vodorovných pruhů. Pro odstranění svislého pruhu byla použita operace uzavření strukturním elementem LINE_X o velikosti 50 (matice 50x1). K odstranění došlo, protože šířka svislého pruhu je méně než 50 pixelů. Obdobně se k odstranění vodorovných použije uzavření strukturním elementem LINE_Y o velikosti 20 (matice 1x20).

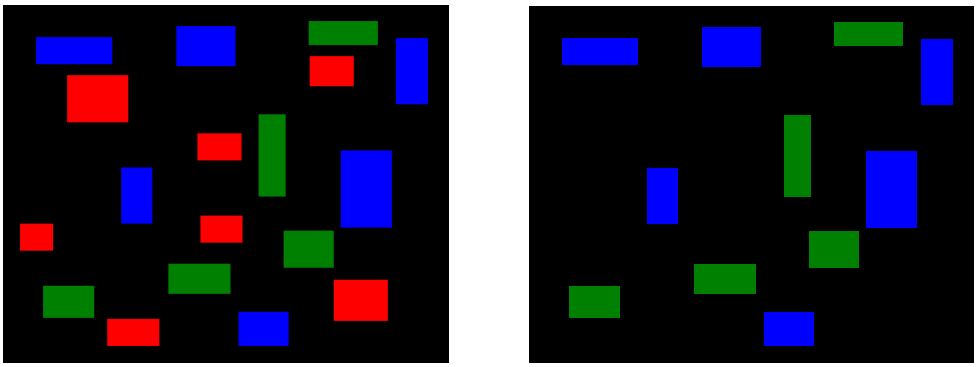
Obrázek 45 – původní obrázek

Obrázek 46 – obrázek s odstraněným svislým pruhem a s odstraněnými vodorovnými pruhy

Barevné obrazy

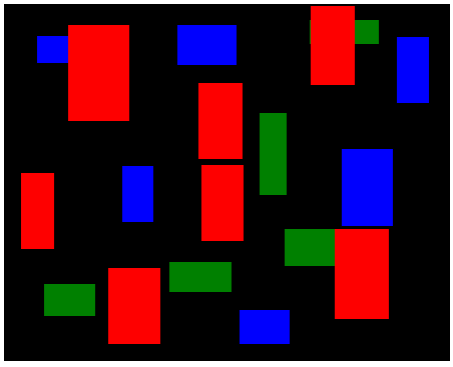
Dalším příkladem je odstranění objektu určité barvy. Zde se jedná o ideální případ, kdy je pozadí černé (všechny tři složky RGB mají nulovou hodnotu) a jednotlivé objekty mají jen jednu nenulovou položku RGB. Potom lze objekt redukovanou erozí podle nenulové položky jeho barvy a použitím vhodného elementu odstranit. U následujícího obrázku byla provedena redukovaná eroze podle položky R modelu RGB strukturním elementem LINE_Y o velikosti 50 (matice 1x50). V reálném příkladě by například bylo možné barevnou položku odprahovat a tak odstranit všechny objekty

s některou z převládajících základních barev (nemusí se jednat pouze o barvu modelu RGB, lze modifikovat i pro jiné modely, kde jednotlivé položky mají jiný význam než barvu).



Obrázek 47 – původní obrázek a obrázek s odstraněnými červenými objekty

Stejně tak lze objekty dané barvy rozšiřovat. Zde byla na obraz aplikována operace dilatace podle položky R modelu RGB strukturním elementem LINE_Y o velikosti 50 (matice 1x50).



Obrázek 48 – obrázek s rozšířenými červenými objekty

4.4 Příklady filtrování obrazu – rozostřování

Jednou z věcí, kterou umí morfologické operátory velice dobře, ale ne často ji vidíme rádi, je rozmazávání obrazu. To, jaký strukturní element a operaci použijeme, je vesměs otázkou vkusu každého z nás. U binárních obrazů je to spíše rozšiřování objektů na úkor pozadí a naopak, zatímco u šedotónových a barevných fotek lze dostat zajímavé výsledky. Opět platí to, že většinou uzavření zesvětluje a otevření ztmavuje.

Šedotónové obrazy

Na následující dvě šedotónové fotografie byl použit filtr uzavření-otevření čtvercovým strukturním elementem o velikosti 3 [29]. Na třetí fotografii jsem aplikovala jen samotnou operaci otevření, která obrázek rozmazala ztmavila a uzavření, která obrázek rozmazala a zesvětlila [30]. Operace byly provedeny opět čtvercovým strukturním o velikosti 3.

Obrázek 49 – původní fotografie a fotografie rozmazaná filtrem uzavření-otevření

Obrázek 50 – původní fotografie a fotografie rozmazaná filtrem uzavření-otevření

Obrázek 51 – původní fotografie a fotografie rozmazané operacemi otevření a uzavření

Barevné obrazy

Na následujícím výřezu s fotografie je zřejmé rozmazání a zesvětlení, které bylo provedeno aplikací filtru uzavření-otevření strukturním čtvercovým strukturním elementem o velikosti 3 marginálním přístupem v modelu YCbCr. Použitím jiných modelů je výsledek po aplikaci stejné operace hodně podobný tomuto.

Obrázek 52 – původní fotografie a fotografie rozmazaná filtrem uzavření-otevření

Na dalším obrázku je provedena operace otevření redukovaným přístupem podle položky Cr barevného modelu YCbCr čtvercovým strukturním elementem o velikosti 5. Redukované operace podle položek různých modelů davájí často zajímavé a dost odlišné výsledky (na rozdíl od marginálního přístupu).

Obrázek 53 – původní fotografie a fotografie pozměněná operací otevření

4.5 Segmentace

Segmentace je dělení obrazu na rozdílné regiony, které mají určité vlastnosti [4]. Není to nic jiného než to, že jednotlivé pixely obrazu patří do některého z regionů. Segmentace je klíčovým krokem při interpretaci obrazu.

Z matematického hlediska je segmentace obrazu f dělení jeho definičního oboru D_f do n disjunktních neprázdných množin X_1, X_2, …, X_n představujících segmenty takových, že jejich sjednocení tvoří původní obraz.

Návrh algoritmů pro segmentaci obrazu do smysluplných regionů vyžaduje určité předchozí znalosti o objektech v obraze, které mají být rozeznány [13]. Tyto znalosti se týkají vlastností jako tvar, velikost, orientace, šedotónové rozložení a textura. Bohužel, v praxi jen zřídkakdy známe tyto potřebné vlastnosti. Navíc obraz často obsahuje šum nebo jsou jeho jednotlivé části jinak osvíceny. Proto neexistuje žádný univerzální algoritmus pro segmentaci všech obrazů.

Segmentace obrazu pomocí morfologie

Morfologický přístup k segmentaci kombinuje metodu narůstání oblastí a detekci hran [4]. Výsledná transformace se nazývá metoda rozvodí (*watershed transformation*). Morfologie napomáhá segmentovat obrazy částic jak pro binární tak pro šedotónové vstupní obrazy.

Metoda narůstání oblastí pracuje tak, že náhodně zvolené pixely obrazu (tzv. *semínka*), tvoří počáteční regiony obrazu [4]. Regiony se rozrůstají na okolní pixely, je-li splněno zadané kritérium homogenity. Detekce hran je postup, sloužící k nalezení oblastí pixelů, ve kterých se podstatně mění jas. Pokud hranu definujeme jako velkou změnu jasové funkce, bude v místě hrany velká hodnota derivace jasové funkce. Maximální hodnota derivace bude ve směru kolmo na hranu. Kvůli jednoduššímu výpočtu se ale hrany detekují jen ve dvou, resp. ve čtyřech směrech.

Metoda rozvodí

2D šedotónový obraz lze přirovnat k terénu nebo topografickému reliéfu, kde tmavé pixely korespondují s kotlinami a údolími a světlé pixely představují kopce a hřebeny hor [1]. Předpokládejme, že se zajímáme o segmentaci „tmavých“ regionů. Intuitivně jsou rozvodí obrazu tvořeny údolími, které jsou odděleny hřebeny hor. Tato údolí tvoří jednotlivé regiony. Pomyslný terén je postupně zaplavován vodou, přičemž jsou zaplňována údolí s regionálními minimy. Výsledný obraz je rozdělen do povodí oddělených hrázemi. Kvůli šumu a texturám mají obrázky často mnoho regionálních minim.

4.6 Další použití morfologických operací

Vedle filtrování a segmentace se lze s morfologií ve zpracování obrazu setkat i v dalších oblastech, kterými jsou *klasifikace*, *analýza textur* a *geodetické metriky* a *transforma*ce [4].

Klasifikace

Finálním krokem při zpracování obrazu je *klasifikace*. Jedná se o zařazení objektů nalezených v obraze do skupiny předem známých tříd. Techniky klasifikace jsou dvou druhů [4]: bodová (pixel-based) nebo objektová (object-based). V případě klasifikace založené na bodech jsou příznaky k dispozici pro každý bod obrazu. Objektová klasifikace je možná až po extrakci jednotlivých objektů (většinou po segmentaci), které přiřadí do tříd.

Mezi bodové klasifikační techniky patří *watershed-based clustering*, která je neparametrizovanou metodou pro nalezení shluků v multivarietních histogramech multispektrálních obrazů. Další metodou pro klasifikaci je *Subsequent spatial segmentation.* S morfologickou klasifikací se lze setkat například při interpretaci satelitních snímků.

Analýza textur

Textura je vlastnost obrazu, který má opakující se strukturu a vlastní a nevlastní složku. Počet primitiv v obraze s texturou je větší než jeho variabilita (má výrazné statistické vlastnosti). Textura v obraze se muže projevit na více úrovních rozlišení. Pak mluvíme o tzv. *charakteristickém měřítku* textury.

Důležitou operací v oblasti morfologické analýzy textur je *granulometrie* (granulometry), která má v morfologii podobně významnou roli jakou hraje frekvenční analýza ve zpracování obrazu. Dále morfologická analýza textur zahrnuje *morfologickou kovarianci* (morphological covariance), kterou lze definovat pomocí autokorelační funkce. Morfologie lze použít i pro získání *orientace směrových struktur* v obraze (např. otisky prstů).

Geodetické metriky a transformace

Ústředním pojmem geodetických metod je *geodetická vzdálenost* (geodesic distance), na které jsou založené mnohé geodetické transformace. Geodetická vzdálenost $d_X(x,z)$ je délka nejkratší cesty mezi dvěma body x, y za podmínky, že leží uvnitř množiny X. Dalšími důležitými pojmy jsou *geodetický kruh*, *geodetická dilatace* a k ní duální *geodetická eroze*.

5 Aplikace Morfi

Původně jsem vytvořila v rámci semestrálního projektu jednoduchou konzolovou aplikaci s operacemi omezenými jen na morfologické operátory. Aplikace byla napsána v jazyce C a pro práci s obrázky používala knihovnu digilib. V pokračování diplomové práce jsem vytvořila již rozsáhlejší aplikaci s uživatelským rozhraním v Javě, s jejíž implementací a použitím Vás seznámí následující kapitola.

5.1 Java

Původně jsem zamýšlela vytvořit aplikaci s použitím knihovny OpenCV, ta však nepodporuje Javu, ve které jsem chtěla Morfi vytvořit. Bylo možné použít tzv. JNI (Java Native Interface), které umožní připojit ke kódu v javě kód v jiném jazyce ve formě dynamické knihovny. Tím ovšem ztratíme platformní nezávislost, protože dynamická klnihovna je přeložena jen pro určitou platformu. Dříve se toto používalo pro jazyk C, ve kterém byly například matematické výpočty efektivnější než v těžkopádně Javě. Teď už je i Java optimalizovaná a rozdíl není tak velký. Nakonec jsem od OpenCV ustoupila, protože Java má pro práci s obrazem třídu BufferedImage s pro mě dostačujícími operacemi.

Java a Java IDE

Java je objektově orientovaný programovací jazyk, který vyvíjí firma Sun Microsystems. Díky své přenositelnosti je používán pro programy, které mají pracovat na různých systémech počínaje čipovými kartami (platforma JavaCard), přes mobilní telefony a různá zabudovaná zařízení (platforma Java ME), aplikace pro desktopové počítače (platforma Java SE) až po rozsáhlé distribuované systémy pracující na řadě spolupracujících počítačů rozprostřené po celém světě (platforma Java EE).

Obrázek 54 – logo Javy

Java je jazyk interpretovaný - místo skutečného strojového kódu se vytváří pouze tzv. mezikód (bajtkód) [19]. Tento formát je nezávislý na architektuře počítače nebo zařízení. Program pak může pracovat na libovolném počítači nebo zařízení, který má k dispozici interpret Javy, tzv. virtuální stroj Javy - Java Virtual Machine (JVM). Správa paměti je realizována pomocí automatického Garbage collectoru který automaticky vyhledává již nepoužívané části paměti a uvolňuje je pro další použití. To bylo v prvních verzích příčinou pomalejšího běhu programů. V pozdějších verzích Javy nebyl mezikód přímo interpretován, ale před prvním svým provedením dynamicky zkompilován do strojového kódu daného počítače (tzv. just in time compilation - JIT). Tato vlastnost zásadním

způsobem zrychlila provádění programů v Javě, ale výrazně zpomalila start programů. V současnosti se převážně používají technologie zvané HotSpot compiler, které mezikód zpočátku interpretují a na základě statistik získaných z této interpretace později provedou překlad často používaných částí do strojového kódu včetně dalších dynamických optimalizací. Dalšími vlastnostmi Javy jsou jednoduchost, distribuovanost, robustnost, bezpečnost, nezávislost na architektuře, přenositelnost a podpora vícevláknových aplikací.

Minimem pro možnost spuštění javovské aplikace je instalace běhového prostředí jazyka tzv. JRE (Java Runtime Error), které obsahuje JVM (Java Virtual Machine), kompilované knihovny a další. Pro vývoj java aplikací je nutná sada nástrojů tzv. JDK, v němž jsou zahrnuty knihovny, překladač, JVM a dokumentace. příklady atd.

JAVA IDE

Java je specifická v dostupnosti vývojových prostředí tzv. IDE (Integrated Development Environment) [18]. Pro žádný jiný jazyk není dostupné takové množství kvalitních prostředí a přitom zcela zdarma. Mezi IDE pro javu patří převážně v Česku vyvíjené Netbeans, které jsou vývojovým prostředím i pro C++ a Ruby. Dalším IDE pro javu a C++ je Eclipse, který vznikl z Visual Age for Java od IBM a oproti ostatním nepoužívá pro GUI standardní java knihovnu swing, ale vlastní SWT. Posledním z vývojových prostředí pro Javu je JBuilder, který jako jediný není open source. Jeho základní verze je však zdarma. Všechny uvedené prostředí jsou samy napsány v javě, což přináší výhodu jednoduché přenositelnosti a mírně obtěžující pomalosti. Jejich nároky na operační paměť také nejsou z nejmenších (256Mb+). Všechny prostředí jsou také modulární, takže umožňují jednoduché rozšiřování o další vlastnosti a update po internetu.

Pro Javu jsem se rozhodla kvůli správě paměti pomocí Garbage Collectoru. Aplikaci jsem vytvářela v Netbeans IDE 6.0.1, které je velice pěkně uživatelsky propracované, ale na konci, kdy byla aplikace už poměrně rozsáhlá bylo velice pomalé.

GUI

Když Java vznikala, byla její grafická část velice nedokonalá a neumožňovala snadné a systematické využití [20]. Postupně však docházelo k neustálému zlepšování, takže se Java brzy stala vhodnou platformou pro tvorbu grafických aplikací. Grafiku jako takovou a GUI nelze v Javě oddělovat. Existují pohromadě, využívají stejné třídy.

První implementací grafiky byla knihovna zvaná *Abstract Windowing Toolkit (AWT)*. Objevila se hned na začátku (JDK 1.0) a její nepříjemné a nesystematické API přetrvává v Javě dodnes. Něco se stále používá, ale jsou to spíš věci související s obecnou grafikou nikoli s GUI. V dalších verzích bylo

sice API přepracováno a podstatně rozšířeno, ale jiné nevýhodné vlastnosti přetrvaly. Filosofie AWT byla taková, že každá komponenta v Javě měla svůj nativní protějšek v systému. Byl to tedy podobný model, jako dnes používají některé GUI platformy v C/C++. I když se jednalo o abstraktní (platformově nezávislé) rozhraní, přenositelnost byla problematická vzhledem k rozdílným vlastnostem nativní úrovně GUI.

Od verze 1.2 máme v Javě novou grafickou knihovnu zvanou *Java Foundation Classes (JFC)*, známou spíše pod názvem *Swing* (dále již budu používat jen toto označení). Původní koncept byl zavržen, Swing byl vytvořen prakticky kompletně na zelené louce (i když v sobě obsahuje AWT API). Základní vlastnosti lze shrnout do těchto bodů:

- Kompletní implementace v Javě. GUI není napasováno na nativní komponenty, vše je zcela platformově nezávislé.
- Intenzivní využití dědičnosti a kompozice. Komponenty GUI využívají objektové vlastnosti Javy, složitější součásti jsou složeny či zděděny z jednodušších.
- Výrazné využití rozhraní. Různé operace v komponentách GUI (kreslení, editace apod.) se provádějí přes rozhraní. Standardní implementace lze nahrazovat vlastními a měnit tak chování komponent.
- Důsledné oddělení funkcionality od vzhledu GUI (look & feel). Témat vzhledu je několik k dispozici přímo ve standardní knihovně, další si lze vytvořit a použít.
- Oddělení dat od objektů GUI u složitějších komponent (strom, tabulka) pomocí tzv. *modelů*. To opět zlepšuje možnosti přizpůsobení a také opakovanou použitelnost komponent.

Uvedené vlastnosti, pro programátory velice příjemné, mají druhou stránku věci - relativně vysoké nároky na rychlost procesoru a hlavně na paměť. V reakci na to vznikly grafické knihovny, které se snaží tento problém odstraňovat. Nejvýznamější z nich je SWT.

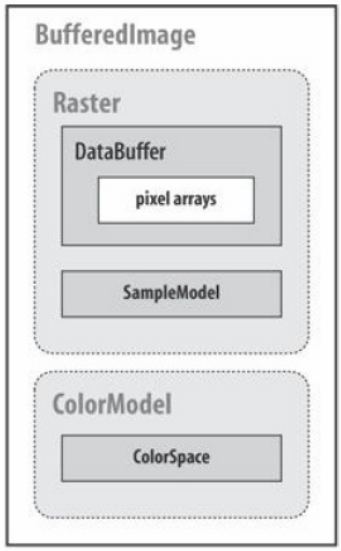
Poslední možností je již zmíněné *SWT (the Standard Widget Toolkit)*, což je GUI knihovna s podobnými vlastnostmi, jako má Swing, ovšem s nativní implementací grafických komponent. K SWT se pojí vývojové prostředí Eclipse. Srovnatelné aplikace postavené na SWT jsou zhruba stejně náročné jako ty používající Swing.

2D API

Java 2D API poskytuje dvou-dimenzionální grafiku, text a možnost zpracování obrazů pro Java programy jako rozšíření AWT [21]. Tento komplexní renderovací balík podporuje čárovou grafiku, text, a obrazy ve flexibilním frameworku pro vývoj bohatších uživatelských rozhraní, sofistikovaných kreslících programů a obrazových editorů. Java 2D objekty existují na ploše nazvané uživatelský souřadnicový prostor. Když jsou objekty renderovány na obrazovce nebo tiskárně, uživatelské

prostorové souřadnice jsou transformovány na prostorové souřadnice zařízení. Důležité třídy pro 2D grafiku v Javě jsou Graphics a z ni zděděná Graphics2D.

V mé aplikaci, kterou lze nazvat obrazovým editorem, jsem mimo Graphics a Graphics2D použila třídu Image, BufferedImage a s nimi související třídy ColorModel, ColorSpace, Raster, WritableRaster a SampleModel. Třída BufferedImage slouží k vytváření bitmapových obrázků, které pak lze v paměti různě upravovat, buď prostřednictvím objektu Graphics, respektive Graphics2D, nebo pomocí objektů Raster. Dále jsem využila možnosti načítání a ukládání formátů obrazů, které umožňuje balík javax.imageio a třídu ImageIO.



Obrázek 55 – struktura třídy BufferedImage

Virtuální paměť javovské aplikace

V závislosti na OS a dostupné paměti RAM existuje limit pro paměť, kterou lze použít. 32-bitový OS a software jsou obvykle limitovány adresováním kolem 2GB až 4GB paměti RAM. To znamená, že pokud je na na počítači nainstalováno 8GB RAM, je možné pro jednu aplikaci stejně použít jen maximálně 2GB. Většina systémů Windows má limit do 2GB, zatímco Mac OS X a Linux může mít přístup až k 4GB na aplikaci. Aktuální 64-bitové OS obchází tato omezení a významně zvyšují hodnotu dostupné paměti.

Standardně přidělená hodnota java aplikaci je 128MB (někde bývá uvedeno 64MB) [22]. Hodnotu lze měnit pomocí parametru **-Xmx** při překladu. Použitím **–Xms** lze nastavit počáteční velikost paměti.

```
java -Xms<initial heap size> -Xmx<maximum heap size>
```

Standardně je použito toto nastavení:

```
java -Xms32m -Xmx128m
```

Samozřejmě jsme omezeni určitou maximální hodnotou. Na 32-bitovém systému Windows s 1GB paměti RAM je limit kolem 1,5GB. Mac OS X a Linux povoluje až 4GB. Obvykle předpokládejme, že je možné alokovat o 50% RAM než je nainstalováno.

Ke zjištění velikosti použité paměti nebo kolik paměti je dostupné, se používá Java Runtime object. Množství alokované paměti pro aplikaci (obvykle –Xms32m nastavení):

```
long allocated = Runtime.getRuntime().totalMemory();
```

Volná paměť (pamět přidělená aplikaci snížena o právě použitou paměť):

```
long free = Runtime.getRuntime().freeMemory();
```

Maximální množství paměti, které případně může tato aplikace spotřebovat:

```
long maximum = Runtime.getRuntime().maxMemory();
```

V Javě verze 6 SE se můžeme setkat s vylepšenou podporou zpracování „out of memory“ výjimek. Výjimka může potom vypadat takto:

```
Exception in thread "main" java.lang.OutOfMemoryError:
Java heap space at
ConsumeHeap$BigObject.(ConsumeHeap.java:22)at
ConsumeHeap.main(ConsumeHeap.java:47)
```

Pokud dojde k takovéto výjimce, může pak jít o dva druhy problémů [23]:

- Java aplikace není schopna si sama ohlídat korektně manipulaci s pamětí a dochází k tzv. paměťovým únikům. Je-li spuštěna, zabírá mnoho místa v paměti a po skončení ji neumí řádně uvolnit. Existují nástroje, které pomohou tyto úniky odhalit, jako např. YourKit Java Profiler.
- Aplikace skutečně potřebuje mnoho paměti (nestačí jí defaultně nastavené maximum). Pak lze limit paměti zvýšit již zmíněným použitím parametru překladu.

Vzhledem k tomu, že vytvořená aplikace pracuje s obrazy, je velmi náročná na virtuální paměť. Při testování často docházelo ke zmíněné „out of memory“ vyjímce. Pomocí programu YourKit Java Profiler jsem zjistila, že nedochází k žádným paměťovým únikům a tak byla výjimka způsobena skutečně nedostatkem paměti při práci s obrazy. Napravila jsem to vymezením více paměti pro aplikaci výše popsaným způsobem.

Překreslování stavového panelu

V programu jsem měla snahu všechny akce zobrazit na stavovém panelu. Avšak narazila jsem na problém, kdy nedošlo k přepsaní informačního panelu (*jLabel*) metodou *setText()*. Nejdříve se provedla akce, při které měl být zobrazen daný nápis (např. Dilating….), a teprve potom se zobrazil nápis. Nepomohlo ani volání metody *validate()* a *repaint()* pro kontejner, kam je daný popisek umístěn. Právě v této situaci, případně ještě pro případ, kdy chceme aby nám např. progressbar zobrazoval průběh operace, se používá více vláken (třída *Thread* nebo rozhraní *Runnable*). Jedno vlákno provádí danou operaci a informuje druhé vlákno, které zobrazuje průběh zpracování. Tato dvě vlákna se střídají buď pravidelně nebo nepravidelně.

Pokud se jedná o pravidelné střídání vláken, pak informační vlákno převezme řízení od pracujícího vlákna vždy, když pracující vlákno zavolá metodu *yield()*. Po vypsání informace o stavu operace informační vlákno předává řízení zpět pracujícímu.

Není-li z povahy algoritmu nutné, aby se vlákna střídala ve své práci pravidelně, je možné realizovat předávání řízení metodami *sleep()* a *yield()*. Pracující vlákno po každém čtení předá řízení pomocí metody *yield()*, zde se tedy nic nezmění. Informační vlákno však pokaždé řízení nepřevezme, protože použije metody *sleep(100)*, která jej vždy uspí na 100 ms (0,1 s) a teprve pak přijme řízení od pracujícího vlákna.

V mém případě bylo ale zbytečné použít progressbar, protože provedení operací je relativně dost rychlé, a tak uživatel nemusí být znepokojen tím „že se dlouho nic neděje“, a stačí jen zobrazit informaci, že se obraz zpracovává. Konečné řešení pomocí anonymní třídy vypadá takto:

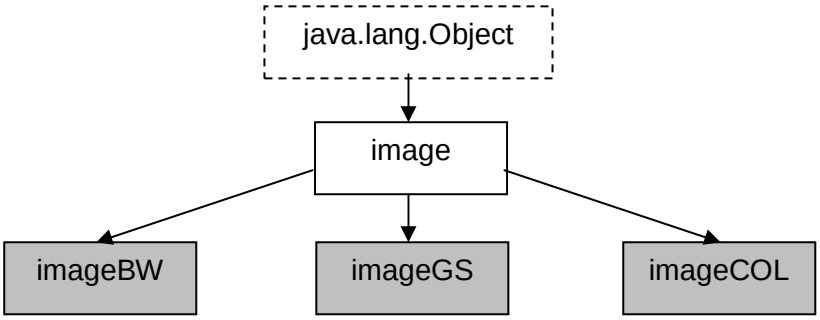
```
new Thread()
            {
              public void run()
              {     //oznámení, že bude prováděna nějaká operace
                    jLabel5.setText("Dilating....");
                   SwingUtilities.invokeLater(new Runnable()
                   {
                     public void run()
                     {//samotné provedení operace dilatace
                     //oznámení, že byla operace dokončena
                     jLabel5.setText("Dilation completed");
                     }
                   });
              }
           }.start();.
```

5.2 Implementace

Při návrhu bylo nutné nejdříve rozhodnout, jaké nové objekty vytvořit, a to tak, aby byly univerzální a dobře se s nimi pracovalo. Vzhledem k tomu, že se jedná o aplikaci pro zpracování obrazu, většina nově vzniklých tříd se vztahuje právě k reprezentaci obrazů.

Hierarchie tříd

Pro funkčnost celé aplikace je nejdůležitější abstraktní třída **image**, která je potomkem třídy **Object**. Potomky třídy **image** pak jsou třídy představující 3 druhy obrazů: **imageBW** pro černobílý, **imageGS** pro šedotónový a **imageCOL** pro barevný obraz.



Obrázek 56 – hierarchie tříd

Dalšími vytvořenými třídami jsou:

ExtensionFilter, MyTableModel, AbstractTableModel, PanelImage, PanelSE, swapImage, swapSE, morfiJFrame.

Třída image (image.java)

Abstraktní třída **image** shrnuje společné vlastnosti obrazů v aplikaci. Jsou v ní konstanty pro typ obrazu, barevný model, strukturní element, pomocné proměnné a další atributy pro uchování vlastností obrazu jako například typ, raster, výška, …

Použité typy obrazu (tak jak jsou v Javě pro *BufferedImage*):

Černobílý - TYPE_BYTE_BINARY (imageBW) Šedotónový - TYPE_BYTE_GRAY (imageGS) Barevný - TYPE_3BYTE_BGR (imageCOL) (představuje tyto typy TYPE_3BYTE_BGR, TYPE_INT_RGB, TYPE_INT_BGR, TYPE_BYTE_INDEXED, TYPE_CUSTOM)

Barevné modely:

RGB, HSB, XYZ, HSL, CMY, YCbCr

Strukturní elementy pro morfologické operace:

```
LINE_X, LINE_Y, SQUARE, DIAMOND, MATRIX_3, MATRIX_5, MATRIX_7,
MATRIX_9
```

Dále je ve třídě **image** definováno několik metod. Část z nich jsou finální metody vracející a nastavující atributy (*getType(),setType(int type),*…). Dalšími jsou metody abstraktní, které jsou použity ve všech potomcích (metody *Dilate(..)*a *Erode(..)*) nebo jen v některých z nich.

Nejdůležitějšími metodami jsou *Dilate()* a *Erode()*. Ty provedou dilataci/erozi obrazu strukturním elementem typu *strElType* o velikosti *length* a výsledek vrátí v *img2*. Pokud je *strElType* uživatelským typem (*MATRIX_X*), pak je strukturní element uložen v poli *array*.

```
public abstract BufferedImage Dilate(BufferedImage img2,int strElType, int length,int[]
array)
public abstract BufferedImage Erode(BufferedImage img2,int strElType, int length,int[]
array)
```

Třída imageBW (imageBW.java)

Nejjednodušším potomkem třídy **image** je **imageBW**. Jak už název napovídá, jedná se o třídu pro černobílé (binární) obrázky. Kromě definice konstruktoru a metod pro dilataci a erozi je zde metoda pro získání obrysu, inverzi, sjednocení a průnik. Pokud není uvedeno jinak, ve všech metodách je *BufferedImage* typu *TYPE_BYTE_BINARY*.

Konstruktor vytvoří binární obraz z obrazu *img* a se zdrojovým souborem *sourceFile*.

```
public imageBW(BufferedImage img, String sourceFile)
```

Metoda *Contour()* najde obrysy objektů v obraze a výsledek vloží do *img2*, který je metodou vrácen.

```
public BufferedImage Contour(BufferedImage img2)
```

Negation() provede inverzi obrazu (množinová operace binární komplement) a výsledek vloží *do img2*, který metoda vrací.

```
public BufferedImage Negation(BufferedImage img2)
```

Metody *Union()* a *Intersection()* provedou operaci binární sjednocení/průnik nad daným obrazem a *img2*. Výsledek uloží do *img2*.

```
public BufferedImage Union(BufferedImage img2)
public BufferedImage Intersection(BufferedImage img2)
```

Třída imageGS (imageGS.java)

Dalším potomkem třídy **image** je třída pro šedotónové obrázky **imageGS**. Opět zde najdeme definice konstruktoru a metod pro dilataci a erozi. Dalšími metodami jsou inverze, sjednocení, průnik a navíc převod šedotónového obrazu na binární. Pokud není uvedeno jinak, ve všech metodách je *BufferedImage* typu *TYPE_BYTE_GRAY*.

Konstruktor vytvoří šedotónový obraz z obrazu *img* a se zdrojovým souborem *sourceFile*.

```
public imageGS(BufferedImage img, String sourceFile)
```

Negation() opět provede inverzi obrazu a výsledek vloží do *img2*, který metoda vrací.

```
public BufferedImage Negation(BufferedImage img2)
```

Gray2BW() převede šedotónový obraz na binární s prahem *treshold*. Výsledek operace je vložen do *img2* typu *TYPE_BYTE_BINARY* a vrácen metodou.

```
public BufferedImage Gray2BW(BufferedImage img2, int treshold)
```

Metody *Union()* a *Intersection()* provedou operaci šedotónového sjednocení/průniku nad daným šedotónovým obrazem a *img2*. Výsledek je uložen do *img2* a vrácen metodou.

```
public BufferedImage Union(BufferedImage img2)
public BufferedImage Intersection(BufferedImage img2)
```

Posledními metodami pro šedotónový obraz jsou *TopHatWhite()* a *TopHatBlack()*. Ty představují operaci „transformace vrchní části klobouku“ pro nalezení světlých (*white*) / tmavých (*black*) objektů v obraze při proměnlivém pozadí. Jako pomocný je opět použit *img2*, který pak metody vrací.

```
public abstract BufferedImage TopHatBlack(BufferedImage img2)
public abstract BufferedImage TopHatWhite(BufferedImage img2)
```

Třída imageCOL (imageCOL.java)

Posledním a nejobsáhlejším potomkem třídy **image** je třída pro barevné obrázky **imageCOL**. Můžeme v ní najít opět definici konstruktoru a morfologických operací dilatace a eroze. Ty jsou zde však dvou druhů – pro marginální a redukovaný přístup. Ve třídě jsou navíc obsaženy metody pro převody obrazů mezi barevnými modely RGB, HSB, XYZ, HSL, CMY a YCbCr a další pomocné metody pro výpočet pixelu v některých barevných modelech.

Metody *Dilate() a Erode()* provádí morfologické operace nad barevným obrazem marginálním přístupem. Jsou univerzální pro všechny barevné modely(RGB, HSB, ..). Pracují se všemi kanály obrazu jako se šedotónovými obrazy, nad těmi je pak provedena dilatace nebo eroze.

```
public BufferedImage Dilate(BufferedImage img2,int strElType, int length, int[] array)
public BufferedImage Erode(BufferedImage img2,int strElType, int length, int[] array)
```

Na rozdíl od předchozích dvou metod provádí *DilateR()* a *ErodeR()* dilataci a erozi nad barevným obrazem redukovaným přístupem. Jsou univerzální pro všechny barevné modely (RGB, HSB, ..). Operace dilatace či eroze (zjišťování maxima či minima) se provádí jen nad jedním kanálem. Výsledek operace je však aplikován na všechny tři kanály. U metod přibyl parametr *plane*, který říká, podle kterého kanálu se má operace provádět.

```
public BufferedImage DilateR(BufferedImage img2,int strElType, int length, int[] array, int colModel, int plane)
public BufferedImage ErodeR(BufferedImage img2,int strElType, int length, int[] array, int colModel, int plane)
```

Následující metody převádí barevný obraz v určitém modelu do jiného barevného modelu. Implementován je vždy převod tam a zpět (z modelu RGB a pak zpět do RGB).

```
public BufferedImage RGBtoHSB(BufferedImage img2 )
public BufferedImage HSBtoRGB(BufferedImage img2 )
public BufferedImage RGBtoXYZ(BufferedImage img2)
public BufferedImage XYZtoRGB(BufferedImage img2)
public BufferedImage RGBtoHSL(BufferedImage img2)
public BufferedImage HSLtoRGB(BufferedImage img2)
public BufferedImage RGBtoCMY(BufferedImage img2)
public BufferedImage CMYtoRGB(BufferedImage img2)
public BufferedImage RGBtoYCbCr(BufferedImage img2)
public BufferedImage YCbCrtoRGB(BufferedImage img2)
```

Ostatní metody třídy slouží jako pomocné pro výpočty při převodech mezi různými barevnými modely.

Další vytvořené třídy

Ostatní třídy již přímo nesouvisí s prováděním morfologických operací, ale slouží pro tvorbu GUI.

1. První třídou je **ExtensionFilter (ExtensionFilter.java)**, která je potomkem třídy **FileFilter**. Umožňuje zobrazení souborů s určitou příponou při otevření dialogového okna pro otevření

nebo uložení obrázku. Třída má dvě proměnné pro ukládání názvů přípon, dva konstruktory a další dvě metody pro testování přípon souborů.

```
public ExtensionFilter(String description, String extension)
public ExtensionFilter(String description, String extensions[])
public boolean accept(File file)
public String getDescription()
```

2. Dále jsem vytvořila třídu **MyTableModel (MyTableModel.java)**, která dědí ze třídy **AbstractTableModel**. Konstruktor zůstává stejný jako u rodičovské třídy, ostatní zděděné metody pro zjištění vlastností tabulky a nastavení buněk jsou předefinovány. Instance této třídy jsou použity v GUI pro zobrazení informací o obrazu.

```
public int getColumnCount()
public int getRowCount()
public String getColumnName(int col)
public Object getValueAt(int row, int col)
public Class getColumnClass(int c)
public boolean isCellEditable(int row, int col)
public void setValueAt(Object value, int row, int col)
```

3. **PanelImage (PanelImage.java)** je třídou, která se používá při zobrazování obrázku. Takto vypadá konstruktor, kde parametr *image je* obraz k zobrazení a *scale* říká, jak velký obraz bude.

```
public PanelImage(image image, double scale)
```

Další veřejná metoda vrací komponentu *JLabel* s obrazem, která je pak vložena do komponenty *JScrollPane*:

```
public JLabel getPanelImage()
```

Poslední metodou, která je soukromá a používá se pro zmenšení či zvětšení obrazu, je:

```
private BufferedImage getScaledInstance(BufferedImage img, double scale, int type)
```

4. Třídou obdobnou předchozí, která se však používá k zobrazení tabulky v panelu pro odkládání strukturních elementů, je **PanelSE (PanelSE.java).** Najdeme v ní proměnné pro uložení strukturního elementu, tabulku či panel. Konstruktor má jako vstupní parametr pouze pole strukturního elementu *SE*.

```
public PanelSE(int SE[])
```

Další metody třídy vrací panel s tabulkou nebo tabulku „naplněné“ strukturním elementem.

```
public JPanel getPanelSE()
public JTable getTableSE()
```

5. Třída **swapImage (swapImage.java)** se používá při odkládání obrázků. Proměnné třídy jsou id, jméno, obraz, položka menu a panel. Konstruktor má parametry identifikační číslo *id* („kolikátý je to odložený obrázek od spuštění aplikace“) a obraz *image* který se „odkládá“. Provede přídání položky do záložek s odloženými obrazy (pro zobrazení náhledu použije třídu **PanelImage**).

```
public swapImage(int id, image image )
```

Další metody slouží k získávání vlastností třídy.

```
public image getImage()
public int getId()
public JLabel getPanel()
public String getName()
public JMenuItem getItem()
```

6. Podobnou funkci jako má pro obraz swapImage, má pro strukturní element třída **swapSE (swapSE.java)** - slouží k odkládání strukturních elementů. Ve třídě jsou definovány konstanty pro typ strukturního elementu, stejné jako jsou ve třídě **image,** a proměnné id, jméno, typ, pole strukturního elementu atd. *Id* v konstruktoru třídy opět značí, kolikátý je to odložený strukturní element od spuštění aplikace. *Array* je pole, které představuje strukturní element, pokud se jedná o uživatelský typ (*MATRIX*), *type* je typ strukturního elementu a *length* je jeho velikost.

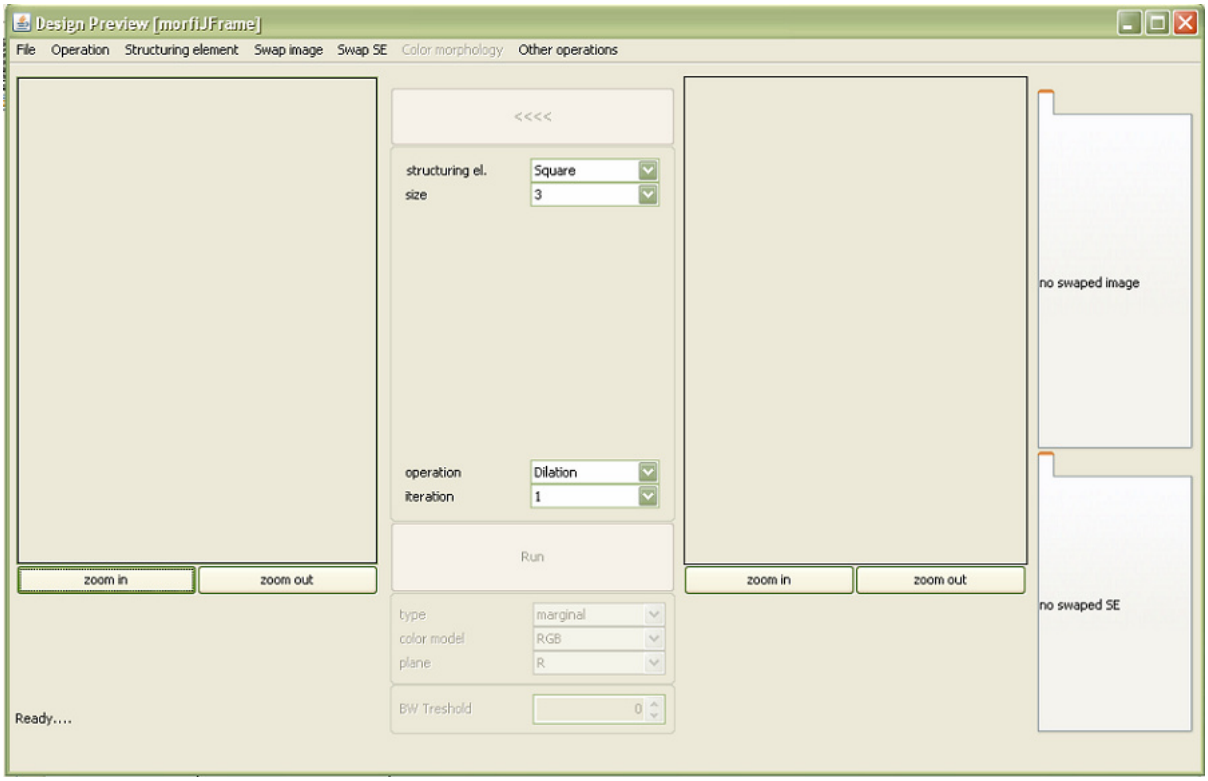
```
public swapSE(int id,int array[],int type,int length)
```

Ostatní metody slouží opět k získávání vlastností třídy.

```
public int getId()
public int getType()
public int getLength()
public int[] getSE()
public JMenuItem getItem()
public JPanel getPanel()
public JTable getTable()
public String getName()
```

Hlavní třída MorfiJFrame

Hlavní třídou aplikace Morfi je **MorfiJFrame (MorfiJFrame.java, MorfiJFrame.form)**. Obsahuje kód uživatelského rozhraní - všechny komponenty, jejich inicializaci, akce k nim příslušící, hlavní metodu a další pomocné metody. Aplikace je řízena událostmi a je použit model asynchronního programování.



Obrázek 57 – náhled GUI aplikace

Nalezneme zde stejné konstanty jako ve třídě **image**. Dalšími konstantami pak jsou:
Typy barevné morfologie

```
MARGINAL, REDUCED
```

Morfologické operace

```
DILATION, EROSION, OPEN, CLOSE
```

Třída obsahuje proměnné pro dva panely s obrázky (jejich měřítko, typ, proměnné pro uložení obrazu a zdrojový soubor), strukturní element (typ, velikost a pole pro uživatelský typ), operace (typ a počet iterací), „odložený“ obraz (počet a pole odložených obrazů), „odložený“ strukturní element (počet a pole odložených strukturních elementů), barevnou morfologii (typ, barevný model a položka), adresář pro otevírání obrazů, tabulky informačních panelů a převod z šedotónového na binární obraz (spinner model a práh).

Všechny komponenty a některé proměnné jsou inicializovány v konstruktoru třídy.

```
public morfiJFrame()
```

Dále třída obsahuje velký počet metod pro obsluhu událostí. Z nich popíši ty nejdůležitější a nejobsáhlejší.

1. První z nich je metoda pro obsluhu události načtení obrázku – *File/Open* nebo kontextové menu panelů s obrázky.

```
private void jMenuItem1ActionPerformed(java.awt.event.ActionEvent evt)
```

   Metoda zobrazí dialogové okno pro výběr souboru k otevření - defaultně je nastaveno zobrazení JPG souborů. Java nativně pracuje s formáty JPG, PNG a GIF a BMP. Původně jsem chtěla, aby aplikace navíc načítala i format TIFF, kvůli jeho bezztrátové kompresi. Bylo by však nutné použít JAI (Java Advanced Imaging) API nebo nějak jinak rozšířit aplikaci. Nakonec jsem se rozhodla, že ponechám jen podporované formáty, přičemž jsem ještě musela ubrat formát GIF, protože z něj načtený *BufferedImage* má jiné vlastnosti než po načtení z jiných formátů. Načítat lze obrazy těchto typů (tak jak jsou definovány pro *BufferedImage*):

   - Binární (*TYPE_BYTE_BINARY*)
     Reprezentuje dvoubarevný obrázek, kde je 1 pixel reprezentován 1, 2 nebo 4 bity. Obrázek je bez kanálu alfa. (Tento typ je pro zpracování převeden na šedotónový 8-bitový obrázek, kde 0 zůstane 0 a 1 je nahrazena 255.)

   - Šedotónový (*TYPE_BYTE_GRAY*)
     Reprezentuje obrázek s odstíny šedi (neindexovaný), kde je 1 pixel reprezentován 1 (bezznaménkovým) bytem.

   - Barevné:
     TYPE_3BYTE_BGR 8-bitů pro každou z komponent BGR (modrá, zelená, červená), 1 pixel je tedy reprezentován 24bity. Nemá alfa složku.
     TYPE_INT_RGB, TYPE_INT_BGR 8-bitů pro každou z komponent RGB, BGR (modrá, zelená, červená), 1 pixel je reprezentován číslem typu integer. Nemá alfa složku.
     TYPE_BYTE_INDEXED reprezentuje indexovaný obrázek (s paletou), kde jeden pixel je uložen na jednom bytu.
     TYPE_CUSTOM – Typ nebyl rozpoznán. (Původně jsem tento typ nechtěla zahrnout, ale jsou v něm uloženy téměř všechny barevné PNG obrázky.)

Uložení obrázku je řešeno podobným způsobem a podporovány jsou stejné formáty.

2. Nejdůležitější komponentou je tlačítko s nápisem „*Run*“, které obsluhuje metoda :

```
private void jButton7ActionPerformed(java.awt.event.ActionEvent evt)
```

Ta vezme obrázek z levého panelu, provede nad ním morfologickou operaci, která je nastavená a výsledný obraz zobrazí na pravý panel.

Morfologické operace nad barevným obrazem s marginálním přístupem jsou provedeny takto:

- Obraz v levém panelu je vždy v modelu RGB.
- Pokud je nastavena operace v jiném než RGB modelu, je obraz převeden do daného modelu.
- Nad obrazem se provede nastavená marginální operace.
- Obraz je převeden zpět do RGB a zobrazen.

Redukovaný přístup řeší barevné převody přímo v metodách třídy **imageCOL**. U této metody obsluhující tlačítko *Run* jsem musela použít pro aktualizaci stavového panelu jednoduchá vlákna.

3. V programu lze odkládat obrázky pro pozdější použití buď z kontextové nabídky obrázků nebo přes nabídku *Swap image/Add image* (odloží obrázek v pravém panelu). Implementace této akce je např. v metodě *jMenuItem205ActionPerformed().*

```
private void jMenuItem205ActionPerformed(java.awt.event.ActionEvent evt)
```

Pří odložení obrázku se přidá nový *swapImage* do pole *swapImages*. Pod položku Add Image v menu Swap Image přibyde nová položka s pořadovým číslem odloženého obrázku a položka s náhledem obrázku přibude také do komponenty *JTabbedPane* pro odložené obrázky, která je vpravo nahoře. Jak k přidané položce v menu, tak položce v *JTabbedPane* jsou přidány akce po stisknutí položky respektive zobrazení kontextového menu.

4. Obdobně je v programu řešeno odložení strukturního elementu (*Swap SE/Add SE*) a to např. v metodě *jMenuItem358ActionPerformed().*

```
private void jMenuItem358ActionPerformed(java.awt.event.ActionEvent evt)
```

Kromě metod pro obsluhu události a inicializace komponent má hlavní třída také pomocné metody.

1. Nejvolanější z nich je metoda *paint()* sloužící k překreslení obrázku v panelech a aktualizaci informačních tabulek obrázků.

```
public void paint(Graphics g)
```

2. Další slouží k práci s polem tlačítek strukturního elementu. K naplnění pole uživatelského strukturního elementu „zmáčknutými“ tlačítky je metoda *buttonsSE()*.

```
public void buttonsSE()
```

3. Po výběru a změně typu strukturního elementu je potřeba změnit nastavení tlačítek, které zajišťuje metoda *setSE()*.

```
public void setSE(int what)
```

Zbylé metody jsou volány při změně typu obrázku v levém panelu, nad kterým jsou prováděny. Je nutné změnit aktivní nabídky např. pro morfologické operace (jiné jsou pro barevný a jiné pro černobílý obrázek) a nebo operace obdobné převodu z šedotónového na binární obrázek.

Zmíněné metody jsou jen zlomek těch, které jsou ve třídě **MorfiJFrame** implementovány. Na ostatní se lze podívat do dokumentace vygenerované pomocí nástroje javadoc nebo přímo do zdrojového kódu, které jsou na přiloženém CD.

6 Závěr

„Morfologické operace ve zpracování obrazu“ jsou velice obsáhlým tématem. Pro zpracování jsem si vybrala jen tu základní část, která však svým rozsahem vystačí na diplomovou práci. V textu se pokouším přiblížit čtenáři (u kterého očekávám alespoň minimální znalosti z počítačové grafiky a zpracování obrazu) použití morfologie ve zpracování obrazu takovým způsobem, aby pochopil nejen samotné základní morfologické operace, ale aby se seznámil i s tím, co je jejich podstatou a jaký je jejich původ. Možnosti užití morfologických operací jsou široké a lze se s nimi setkat v mnoha oblastech zpracování obrazu a vědeckých disciplínách. V práci je popsáno použití morfologických operátorů při filtrování obrazu za účelem odstranění šumu nebo objektů určitých vlastností. Přidána je i možnost použít morfologickou filtraci pro rozostření a rozmazání obrazu. Na konci textu uvádím popis implementace aplikace vytvořené v Javě, jejíž kód a testovací obrázky jsou na přiloženém CD. Tuto aplikaci lze použít k demonstraci základních morfologických operátorů a k jednodušším filtrováním.

Informace k tématu binární a šedotónové morfologie jsem získala převážně z české literatury, přičemž vpodstatě všechny vyhledané texty (na internetu) v českém jazyce čerpají z publikace “Zpracování signálů a obrazů“ od autorů Sedláčka a Hlaváče [1]. Ve vyhledaných česky psaných zdrojích jsem nalezla jen nepatrné zmínky o barevné morfologii, a proto jsem čerpala ze zdrojů v anglickém jazyce. K mnoha pojmům v textu týkajících se morfologie uvádím i anglický název, protože některé z nich nebyly v češtině ještě zavedeny.

Ve mnou dostupných zdrojích byly informace k problematice binární morfologie v podstatě shodné. Jinak tomu bylo u tématu šedotónové morfologie, kde je nejednotnost v pojmenovávání některých pokročilých operací (např. transformace horní části klobouku). V případě barevné morfologie je to co článek, to jiný přístup a rozdělení. Zvolila jsem rozdělení dle mého názoru nejlogičtější. Pro implementaci jsem vybrala marginální a redukovaný přístup. Marginální přístup je zcela základní (přirozený) a většina dalších přístupů je jistou modifikací právě redukovaného přístupu. Při praktickém použití těchto přístupů si navíc lze částečně představit výstupy.

Za přínos této práce lze považovat shrnutí problematiky barevné morfologie v českém jazyce. Užitečné jsou nepochybně i uvedené postupy při filtraci pomocí morfologických operací. Přínosem pro mě jako autorku je hlubší pochopení problematiky zpracování obrazu a osvojení si práce s obrázky a s GUI v programovacím jazyce Java. Jako pokračování této práce si dokážu představit popis a implementaci dalších morfologických operací pro šedotónový obraz a především implementace operace tref či miň a její použití při nalézání skeletu.

Literatura

[1] Hlaváč, V., Sedláček, M.: Zpracování signálů a obrazů. 2. přepracované a rozšířené vydání, Praha: Vydavatelství ČVUT, 2005. ISBN 80-01-03110-1. 255 s.

[2] Žára, J. a spol.: Moderní počítačová grafika: kompletní průvodce metodami 2D a 3D grafiky. 2. přepracované a rozšířené vydání, Brno: Computer press, 2004. ISBN 80-251-0454-0. 609 s.

[3] Kaluža, R.: Digitalizace obrazu [online]. 2006 [cit. 2008-05-15]. <http://radovan.bloger.cz/informatika/graficky-orientovane-IS/digitalizace-obrazu>.

[4] Soille, P.: Morphological image analysis. 2. vydání, Berlin: Springer-Verlag, 2003. ISBN 3-540-42988-3. 392 s.

[5] Barevné modely [online]. 2008 [cit. 2008-05-15]. <http://gis.zcu.cz/studium/pok/Materialy/Book/ar03s01.html>.

[6] Wikipedia: HSV [online]. 2008, [cit. 2008-05-15]. <http://cs.wikipedia.org/wiki/HSV>.

[7] Hlaváč, V.: Matematická morfologie [online]. [cit. 2008-05-15]. <http://cmp.felk.cvut.cz/~hlavac/Public/TeachingLectures/BinMatMorfolCesky.pdf>.

[8] Hlaváč, V.: Šedotónová matematická morfologie [online]. [cit. 2008-05-15]. <http://cmp.felk.cvut.cz/~hlavac/TeachPresCz/11DigZprObr/71-06SedaMatMorfolCesky.pdf>.

[9] Zobecněné morfologické operace [online]. [cit. 2008-05-15]. <http://www.dbme.feec.vutbr.cz/ubmi/courses/MASO/MASO14.pdf>.

[10] Köppen, M. a spol.: Fuzzy-subsethood based color image processing [online]. [cit. 2008-05-15]. <http://visionic.fhg.de/ipk/publikationen/pdf/fns99slides.pdf>.

[11] Comer, L. M. a spol.: Morphological operations for color image processing [online]. 1999 [cit. 2008-05-15]. < ftp://skynet.ecn.purdue.edu/pub/dist/delp/color-morph/paper.pdf>.

[12] Třešňák, K.: Barvy a barevné modely [online]. 2001 [cit. 2008-05-15]. <http://www.printing.cz/art/colormanagement/barvy_a_modely.html>

[13] Jain, R., Kasturi, R., Schunck, B. G.: Machine vision. 1. vydání, New York : Osborne-McGraw-Hill, 1995. ISBN 0-07-032018-7 . 549 s.

[14] Čada, R. a spol.: Diskrétní matematika [online]. 2004 [cit. 2008-05-15]. <http://martin.lipinsky.cz/skola/dma/prednasky/0.pdf>.

[15] Wikipedia: Duality (Mathematics) [online]. 2008 [cit. 2008-05-15]. <http://en.wikipedia.org/wiki/Duality_%28mathematics%29>.

[16] Málek , M.: Základy topologie [online].2004 [cit. 2008-05-15]. <http://homepages.math.slu.cz/MichalMalek/vyuka/04-05/Zaznamy-zima/Topologie.pdf>.

[17] Homeomorfismus [online]. 2008 [cit. 2008-05-15]. <http://wapedia.mobi/cs/Homeomorfismus>.

[18] Patrný, V.: Java IDE [online] 2003 [cit. 2008-05-15]. <http://www.linuxzone.cz/index.phtml?idc=741&ids=6>.

[19] Wikipedia: Java [online]. 2008 [cit. 2008-05-15]. <http://cs.wikipedia.org/wiki/Java>.

[20] Jelínek, L.: Java (24) – úvod do grafiky [online]. 2006 [cit. 2008-05-15]. <http://www.linuxsoft.cz/article.php?id_article=1184>.

[21] Overview of the Java 2D API concepts [online]. 2006 [cit. 2008-05-15]. <http://dione.zcu.cz/java/docs/tutorial/2d/overview/index.html>.

[22] Processing : Out of memory errors (java.lang.OutOfMemoryError) [online]. [cit. 2008-05-15]. <http://processing.org/reference/troubleshooting/index.html#memory>.

[23] Hausheer, D.: Increasing heap size in Java to prezent java.lang.OutOfMemoryError [online]. 2007 [cit. 2008-05-15]. <http://hausheer.osola.com/docs/5>.

[24] Binary morphological operators [online]. [cit. 2008-05-15]. <http://documents.wolfram.com/applications/digitalimage/UsersGuide/Morphology/ImageProcessing6.3.html>.

[25] Odstranění šumu a segmentace obrazu pomocí binární morfologie [online]. [cit. 2008-05-15]. <http://cmp.felk.cvut.cz/~svoboda/Vyuka/ZSOZS9900/cvic6.html>.

[26] Noise generation [online]. 2006 [cit. 2008-05-15]. <http://homepages.inf.ed.ac.uk/rbf/HIPR2/noise.htm>.

[27] Filtering salt-and-pepper noise [online]. [cit. 2008-05-15]. <http://www.mmorph.com/handson/handson/fig/hofig2s10.html>.

[28] Pelikán, J.: úloha 54- Filtrace šumu [online]. [cit. 2008-05-15]. <http://cgg.ms.mff.cuni.cz/~pepca/lectures/cv/54noisefilter.html>.

[29] Norman, D.A.: New york art photography Manhattan Bridge [online]. 2007 [cit. 2008-05-15]. <http://www.flickr.com/photos/brooklynlens/>.

[30] Davies, J.: Stockport viaduct, England [online]. 1986 [cit. 2008-05-15]. <http://www.thinkcamera.com/news/article/mps/uan/593>.

Seznam příloh

Příloha 1. Uživatelský manuál k aplikaci Morfi.

Příloha 2. CD obsahující tuto práci, zdrojové kódy a sadu testovacích obrázků pro aplikaci.

Příloha - Uživatelský manuál k aplikaci

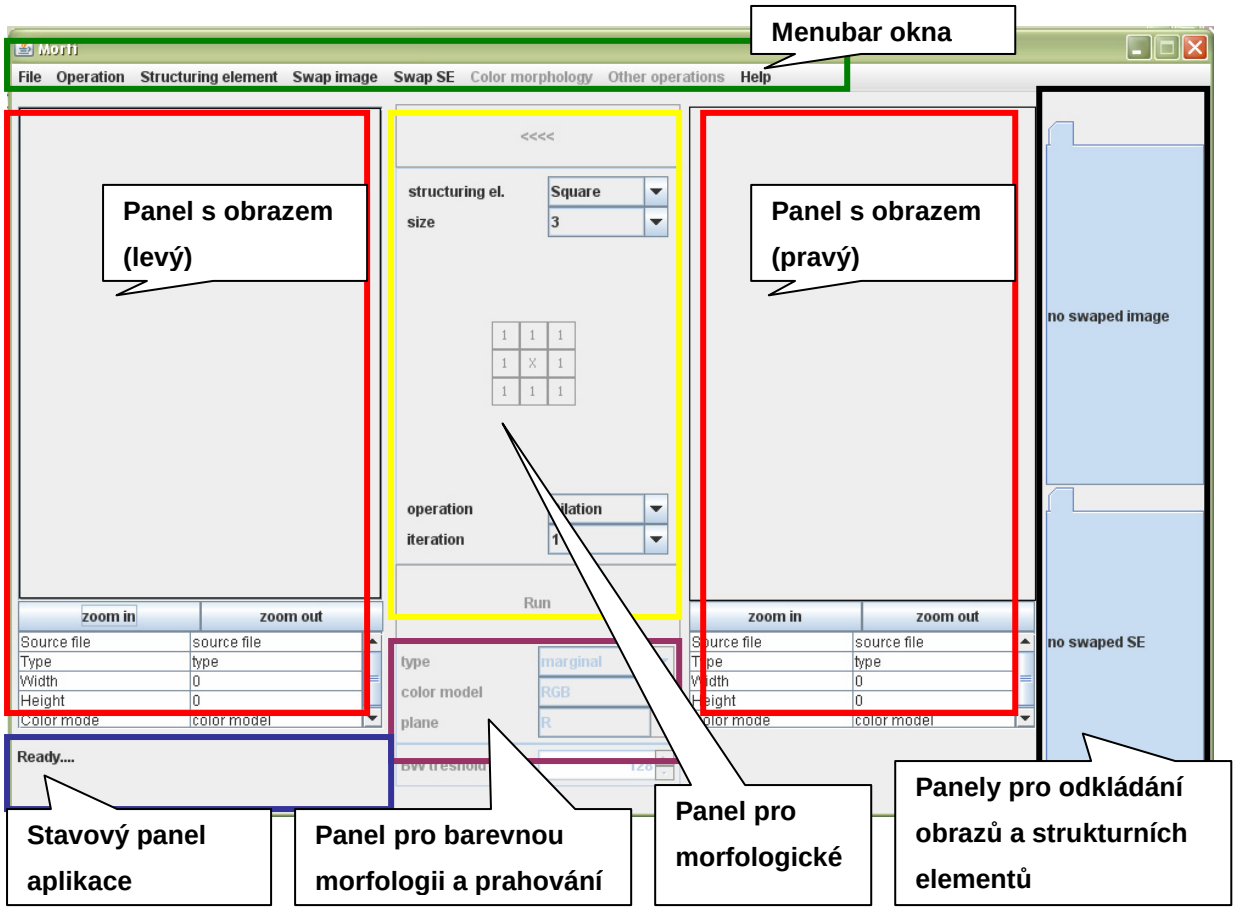
O aplikaci Morfi

Morfi je aplikace pro demonstraci morfologických operací dilatace, eroze, otevření a uzavření nad všemi druhy obrazů (binární, šedotónové, barevné). Pro morfologické operace lze použít strukturní elementy různých druhů a velikostí. K barevné morfologii lze přistupovat marginálním nebo redukovaným přístupem a to v barevných modelech RGB, HSB, XYZ, HSL, CMYa YCbCr. Pro úplnost jsou do aplikace přidány i některé nemorfologické operace jako např. převody mezi barevnými prostory, množinová sjednocení, průniky, komplementy a rozdíly.

Spuštění

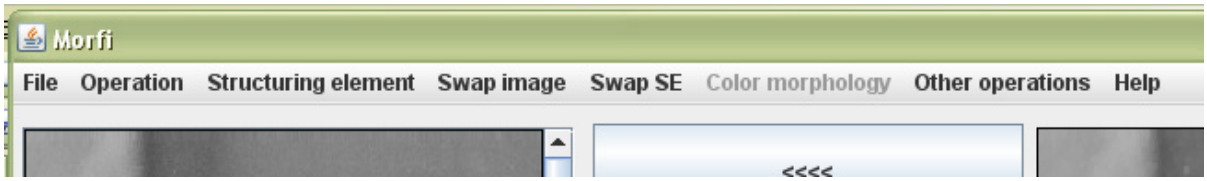
Aplikace **Morfi** nevyžaduje žádnou instalaci. Pro spuštění je pouze nutné mít nainstalované běhové prostředí jazyka Java (JRE – Java Runtime Environment) ve verzi 6.0. To zdarma získáte ze stránek www.java.com. Co se týče velikosti okna aplikace, předpokládá se rozlišení obrazovky 1024x768 pixelů a více.

Popis okna aplikace

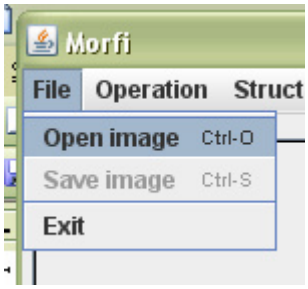


Menubar

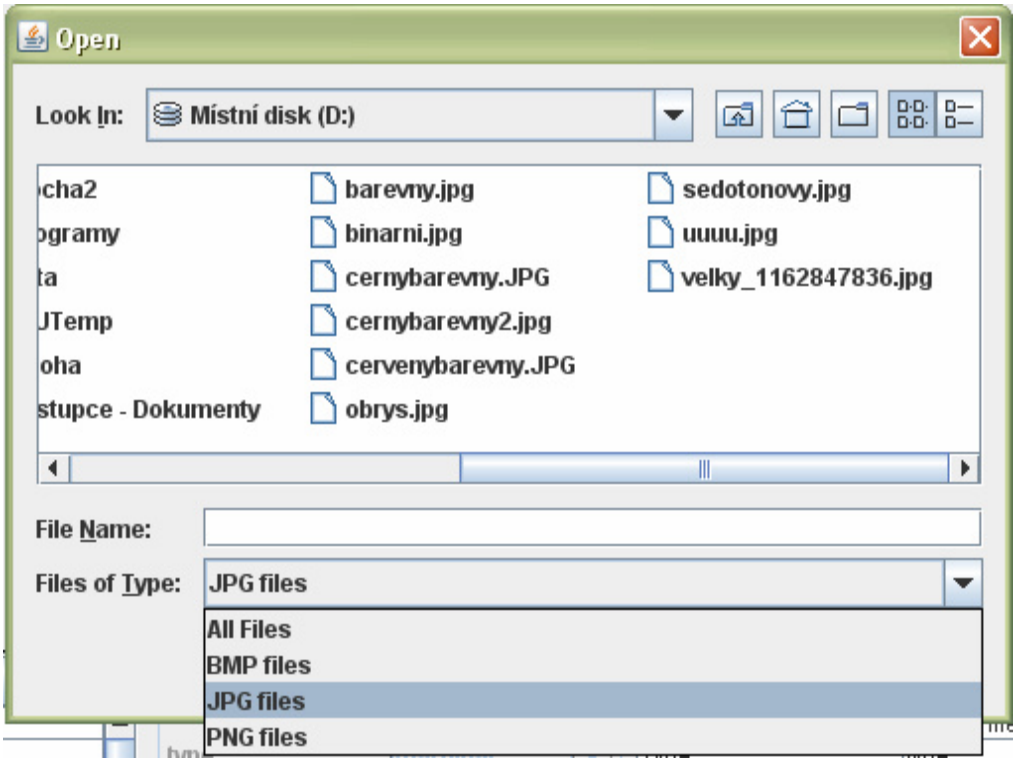
Takto vypadá lišta s jednotlivými menu v horní části aplikace:



1. Menu **File** obsahuje položky **Open image**, **Save image** a **Exit**.

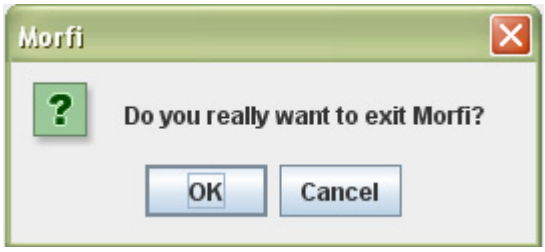


- **Open image (Ctrl+0)** otevře dialogové okno pro výběr obrázku k otevření. Aplikace může pracovat s obrázky formátů jpg (jpeg), bmp a png. Načtený obrázek se zobrazí v levém panelu.

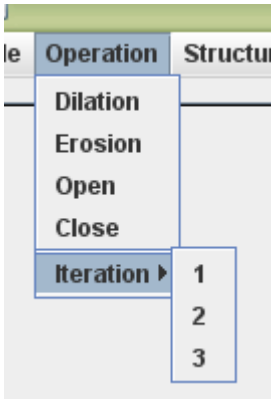


- **Save image (Ctrl+S)** otevře dialogové okno pro uložení obrázku. Ukládá se obrázek v pravém panelu. Formáty pro ukládání jsou stejné jako pro otevírání obrázků.

- **Exit** otevře dialogové okno dotazující se, jestli opravdu chcete ukončit aplikaci.



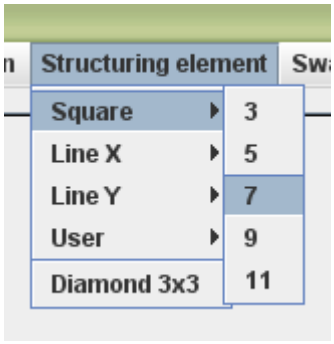
2. Menu **Operation** umožňuje nastavit morfologickou operaci, která se bude provádět a počet iterací.



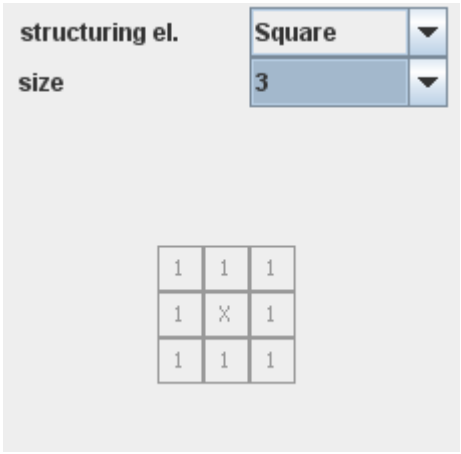
- Výběrem položky **Dilation, Erosion, Open** nebo **Close** zvolíme morfologickou operaci, která se bude provádět. Alternativně lze vybrat operaci i v roletové nabídce panelu pro morfologické operace. V této nabídce se projeví i výběr položky tohoto menu.

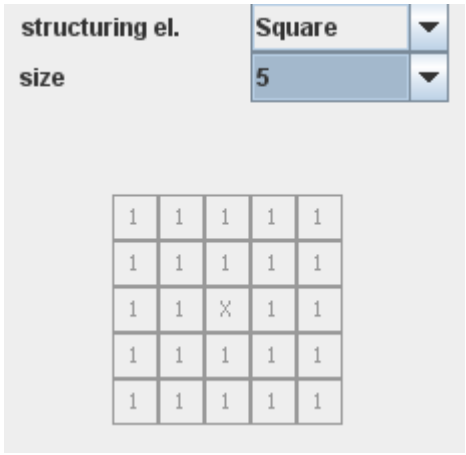
- Výběrem položky v menu **Iteration** nastavíme počet opakování aktuálně nastavené morfologické operace. Opět lze zadat počet iterací i v roletové nabídce v panelu pro morfologické operace, kde je možné nastavit více než 3 iterace.

3. Menu **Structuring element** umožňuje nastavit strukturní element pro provedení morfologické operace. Výběru (s větším rozsahem velikostí), který lze provést prostřednictvím tohoto menu, lze docílit i v panelu morfologických operací. (Ve všech strukturních elementech je znak X jejich počátkem.)

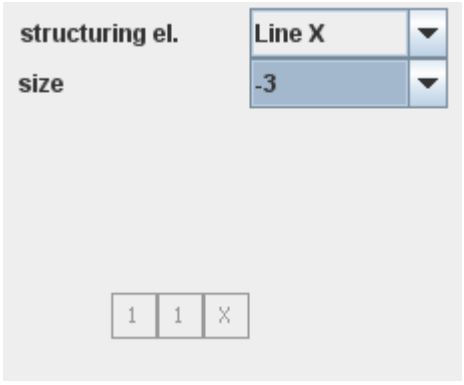


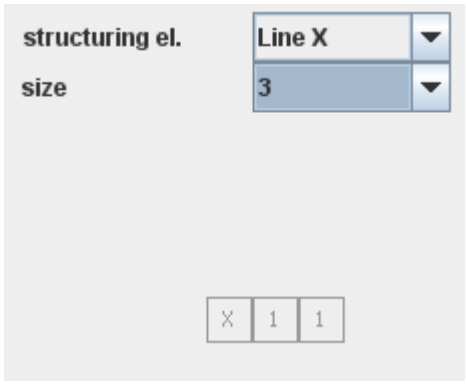
- **Square** představuje čtvercový strukturní element o velikosti 3x3, 5x5, 7x7, 9x9, 11x11, 13x13 nebo 15x15 pixelů.



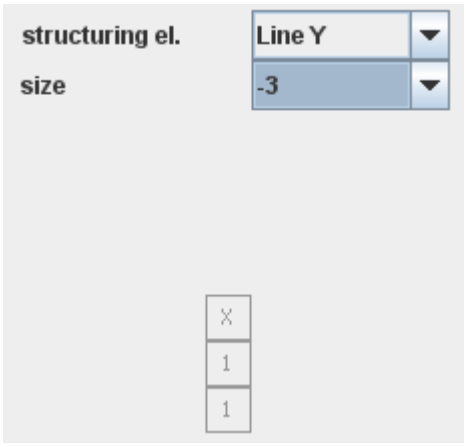


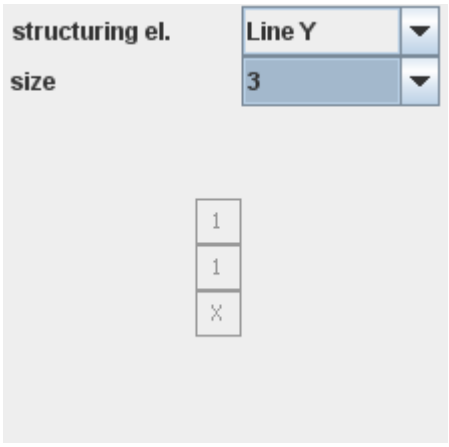
- **Line X** je strukturní element představující matici o jednom řádku a dvou a více sloupcích, u kterého lze v této nabídce zvolit velikosti -3, -2, 2, 3 a 4. *(V roletové nabídce je možné vybrat i jiné rozměry.)*



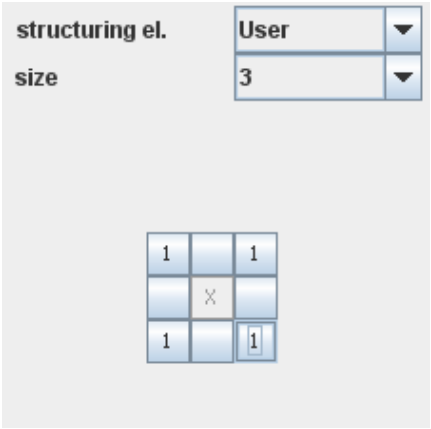


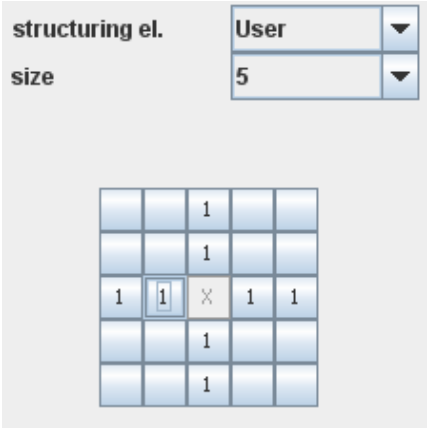
- **Line Y** je strukturní element představující matici o jednom sloupci a dvou a více řádcích, u kterého lze v této nabídce zvolit velikosti -3,- 2, 2, 3 a 4. *(V roletové nabídce je možné vybrat i jiné rozměry.)*





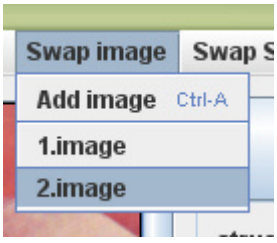
- **User** představuje uživatelský strukturní element. Vybrat lze čtvercovou matici o velikostech 3x3, 5x5, 7x7 a 9x9 pixelů. Samotné pixely matice lze nastavovat v poli tlačítek v panelu morfologických operací.



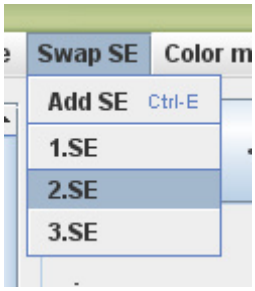


- **Diamond 3x3** je zde zahrnut, protože se jedná o jeden z nejpoužívanějších strukturních elementů.

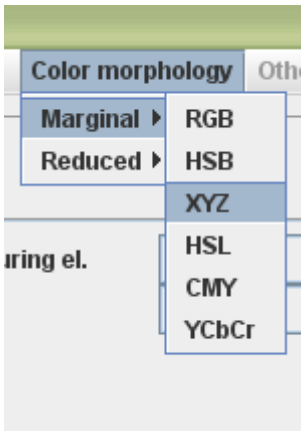
4. Menu **Swap image** po spuštění aplikace obsahuje pouze položku **Add Image (Ctrl+A)**, která provede odložení obrázku v pravém. Do panelu **Swap image** je přidána položka s pořadovým číslem odloženého obrázku a v panelu pro odložené obrázky (vpravo nahoře) přibude také nová položka. Po výběru položky nějakého obrázku z menu (např. *1.image*) je tento obrázek načten do levého panelu a je možné s ním dále pracovat. To stejné lze provést i poklepáním levého tlačítka myši na náhled odloženého obrázku v panelu pro odložené obrázky.

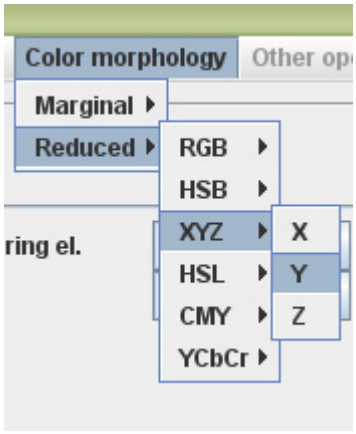


5. Menu **Swap SE** po spuštění aplikace obsahuje pouze položku **Add SE (Ctrl+E)**, která provede odložení aktuálního strukturního elementu. Do panelu **Swap SE** je přidána položka s pořadovým číslem odloženého strukturního elementu a v panelu pro odložené strukturní elementy (vpravo dole) přibude také nová položka. Po výběru položky nějakého SE z menu (např. *1.SE*) je tento SE nastaven v poli tlačítek a v roletových nabídkách v panelu morfologických operací a je možné ho opět použít. To stejné lze provést i poklepáním myší na náhled odloženého SE v panelu pro odložené SE.



6. Menu **Color morphology** je aktivní pouze je-li v levém panelu barevný obrázek. Alternativně lze nastavit tyto operace i v panelu pro barevnou morfologii dole uprostřed pomocí roletových nabídek.

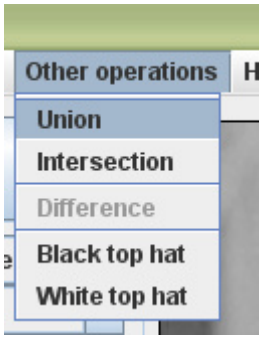




- Menu **Marginal** nastaví marginálního přístupu pro barevnou morfologii s položkami představujícími barevné modely, ve kterých lze morfologické operace provádět.

- Menu **Reduced** provede nastavení redukovaného přístupu s položkami představujícími barevné modely a podpoložkami pro výběr kanálu, podle kterého se bude morfologická operace provádět.

7. Menu **Other operations** je aktivní pouze když jsou v levém a pravém panelu současně binární nebo šedotónové obrázky.



- Položky **Union** a **Intersection** provedou nad binárními/šedotónovými obrazy operace binárního/šedotónového sjednocení nebo průniku. Výsledek operace se zobrazí v pravém panelu.

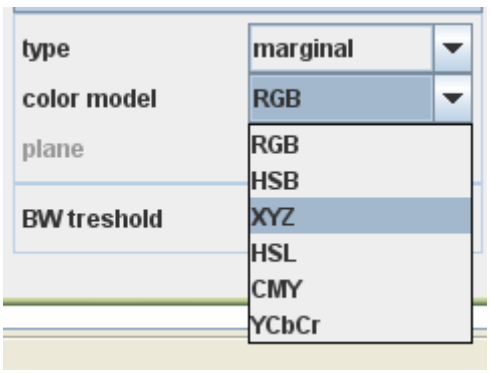
- **Difference** je aktivní pouze v případě, že jsou oba obrazy v panelech binární a provede nad nimi operaci množinového rozdílu. Výsledek operace se opět objeví v pravém panelu.

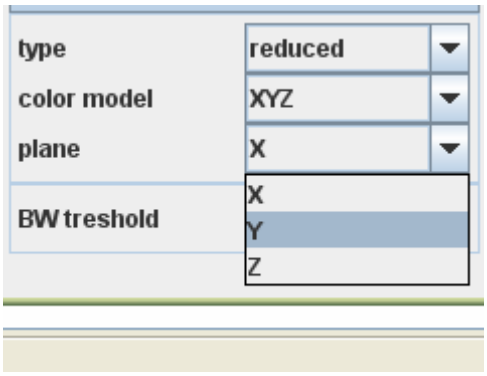
- **Black top hat**/**White top hat** jsou aktivní, pokud je v levém panelu načten šedotónový obraz. Provedou transformaci vrchní části klobouku pro zvýraznění tmavých/světlých míst obrazu. Výsledný obraz se objeví v pravém panelu.

8. Menu **Help** je posledním v horní části okna. Obsahuje jen 2 položky. **Help** otevře nápovědu pro práci s aplikací a **About** zobrazí dialogové okno s informacemi o aplikaci.

Panel pro barevnou morfologii a prahování

Panel pro barevnou morfologii je aktivní pouze pokud je v levém panelu barevný obraz.

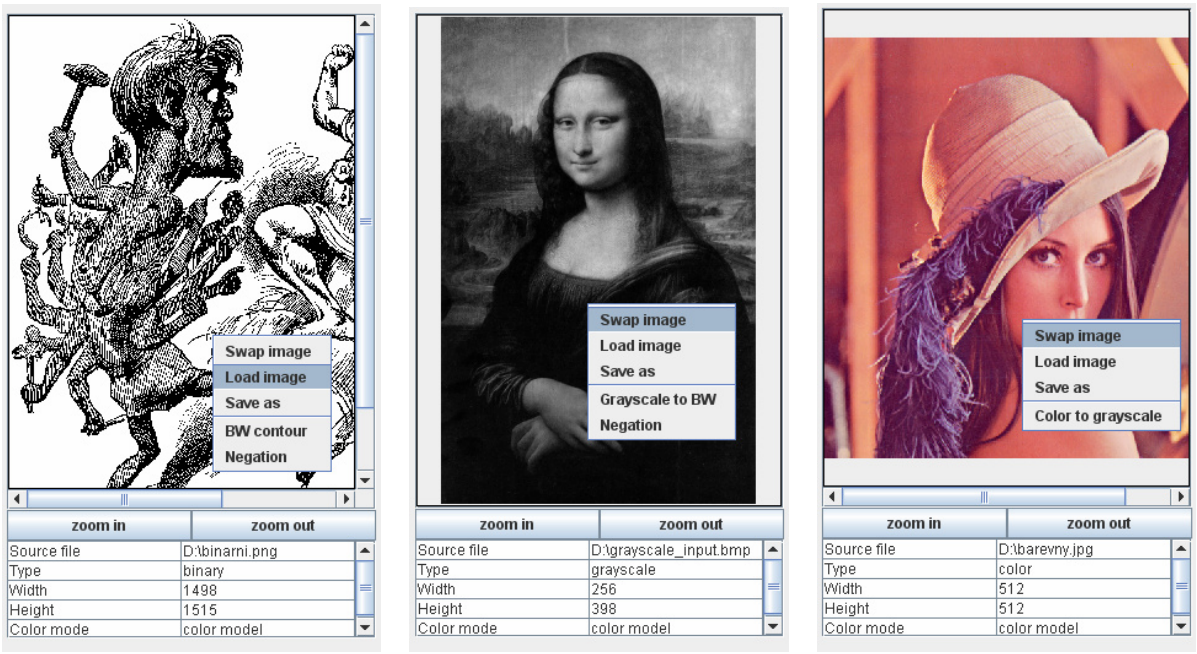




1. V rolovací nabídce **type** lze vybrat, jestli použít marginální nebo redukovaný přístup a v **color model** určíme to, v jakém barevném modelu se má operace provést. Pokud je vybrán redukovaný přístup, je aktivní i nabídka **plane**, ve které se volí, podle kterého kanálu barevného modelu se bude operace provádět.

2. Pokud je v levém nebo pravém panelu šedotónový obraz, je aktivní spinner **BW treshold**, který udává práh pro převod šedotónového na binární obraz.

Panely s obrazy



1. **Kontextové menu obrazu** se zobrazí po kliknutí pravým tlačítkem myši na obrázek v levém i v pravém panelu. Pro různé druhy obrazu se zobrazí různé nabídky. Společné pro všechny obrazy jsou položky **Swap image**, **Load image** a **Save as**.

   - **Swap image** provede odložení obrazu pro pozdější použití. Přidá se položka do menu **Swap image** také do panelu s odloženými obrazy.
   - **Load image** zobrazí dialogové okno a umožní do panelu načíst nový obraz ze souboru.
   - **Save as** zobrazí dialogové okno pro uložení obrázku.

   Binární obraz má v nabídce navíc položky **BW contour** a **Negation**.

   - **BW contour** najde v binárním obraze obrys objektu s použitím právě zvoleného strukturního elementu.
   - **Negation** provede inverzi binárního obrazu.

U šedotónového obrazu jsou navíc položky **Grayscale to BW** a **Negation**.

- **Grayscale to BW** převede šedotónový obraz na binární. Jako práh je použita hodnota ze spinneru **BW treshold** v prostředním panelu úplně dole.
- **Negation** provede inverzi šedotónového obrazu.

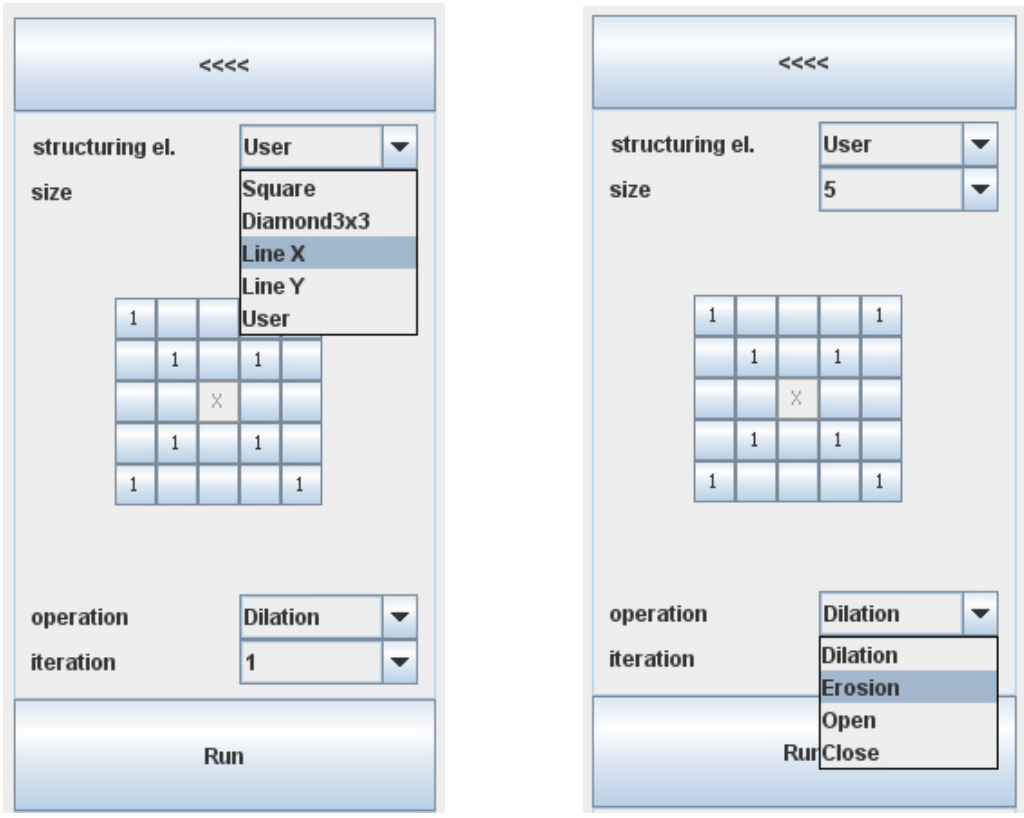
Barevný obraz má v nabídce navíc pouze jednu operaci **Color to grayscale**, která převede barevný obraz na obraz se stupni šedi.

2. Tlačítka **zoom in** a **zoom out** pod obrazem umožňují jeho zvětšení nebo zmenšení.
3. V tabulce dole jsou zobrazeny informace o obrazu načteném v panelu.

Stavový panel aplikace

Zde jsou zobrazovány informace o právě prováděné morfologické operaci nebo např. o úspěšném či neúspěšném uložení obrazu.

Panel pro morfologické operace



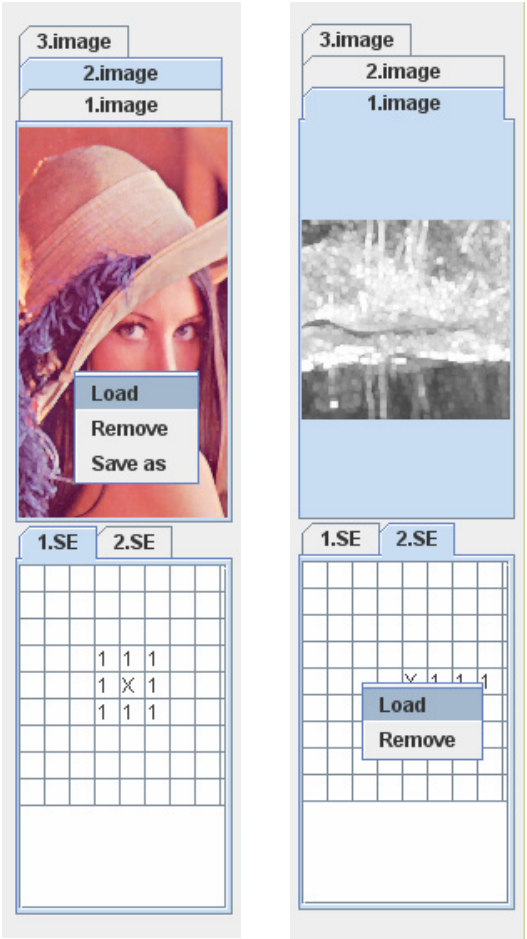
1. Tlačítko <<<< slouží k přesunu obrazu z pravého do levého panelu tak, aby s ním bylo možné dále pracovat.

2. V roletové nabídce **structuring el.** lze vybrat strukturní element pro morfologickou operaci a v nabídce **size** jeho velikost. Pokud je vybrán uživatelský strukturní element (*User*), lze v poli tlačítek zvolit jeho jednotlivé pixely.

3. Roletová nabídka **operation** umožňuje zvolit morfologickou operaci a **iteration** počet opakování této operace.

4. Tlačítko **Run** provede morfologickou operaci (zvolenou v prostředním panelu) nad obrazem vlevo a výsledek zobrazí do pravého panelu.

Panely pro odkládání obrazů a strukturních elementů



Do panelů lze přidávat obrazy a strukturní elementy výběrem menu **Swap image(SE)/Add image(SE)**. U obrazů je možné ještě použít kontextovou nabídku obrazu (položka **Swap image**).

Po stisku pravého tlačítka myši na odložený obraz nebo strukturní element se zobrazí menu s položkami:

- **Load** umožňuje načíst vybraný odložený obraz do levého panelu respektive vybraný strukturní element do panelu s morfologickými operacemi.
- **Remove** odstraní z panelu danou položku (obraz nebo strukturní element).
- U odložených obrazů je navíc v nabídce položka **Save as**, která zobrazí dialogové okno pro uložení obrazu.