VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

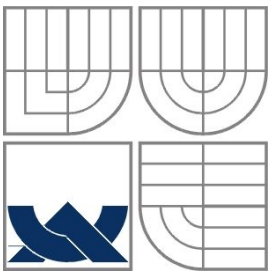
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KONVERZE RASTRU NA VEKTOR

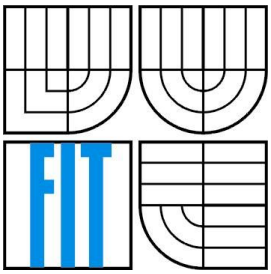
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JAN SIBLÍK
AUTHOR

BRNO 2007



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

KONVERZE RASTRU NA VEKTOR

RASTER-VECTOR CONVERSION

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JAN SIBLÍK

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JANA ŠILHAVÁ

BRNO 2007

Zadání bakalářské práce/5707/2006/xsibli00

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Siblík Jan**
Obor: Informační technologie
Téma: **Konverze rastru na vektor**
Kategorie: Počítačová grafika

Pokyny:

1. Prostudujte a popište problematiku týkající se konverze rastrové reprezentace obrazu na vektorovou.
2. Na základě nastudovaných informací navrhněte a popište alespoň dva algoritmy, které převedou rastrový obraz na vektorový (zadání konzultujte s vedoucím).
3. Implementujte vybrané algoritmy na několika příkladech.
4. Demonstrujte funkčnost algoritmů na rastrových obrazech.
5. Zhodnoťte dosažené výsledky.

Literatura:

- Gonzalez, R. C., Woods, R. E.: Digital image processing. Prentice-Hall, 2002.
- Hlaváč, V.: Zpracování signálů a obrazů. Praha, ČVUT, 2005.
- Galbiati, L. J.: Machine vision and digital image processing fundamental. Prentice-Hall, 1990.
- Žára, J., Beneš, B., Sochor, J., Felkel, P.: Moderní počítačová grafika. Computer Press, 2004.
- Dále dle pokynu vedoucího.

Při obhajobě semestrální části projektu je požadováno:

- první dva body

Podrobné závazné pokyny pro vypracování bakalářské práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva bakalářské práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap (20 až 30% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Šilhavá Jana, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Jan Siblík**
Id studenta: 84157
Bytem: Budovatelská 4816, 760 05 Zlín
Narozen: 18. 07. 1985, Zlín
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
bakalářská práce

Název VŠKP: Konverze rastru na vektor
Vedoucí/školitel VŠKP: Šilhavá Jana, Ing.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☒ ihned po uzavření této smlouvy
   - ☐ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

....

Nabyvatel Autor

Abstrakt

Tato bakalářská práce se zabývá problematikou převodu rastrového obrazu do vektorové reprezentace. Popisuje teoretické podklady pro postupy pro předzpracování vstupního obrazu, teorii a možnosti detekce hran a některé možné postupy vektorizace obrazu. Popisuje také návrh a realizaci demonstrační aplikace, která je její programovou částí.

Klíčová slova

Vektorizace, předzpracování obrazu, detekce hran, segmentace

Abstract

This bachelors thesis inspects an issue of converting a raster image into vector representation. It describes theoretical basis for image preprocessing techniques, theory and means of edge detection and some possible techniques of image vectorization. It also desribes designing and realization of a demostrative aplicetion, which is the programming part of this thesis.

Keywords

Vectorization,image preprocessing, edge detection, segmentation

Citace

Jan Siblík: Konverze rastru na vektor, bakalářská práce, Brno, FIT VUT v Brně, 2007

Konverze rastru na vektor

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Jany Šilhavé. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jan Siblík
13. května 2007

Poděkování

Rád bych zde poděkoval vedoucí své bakalářské práce za trpělivost a také svým rodičům za morální i materiální podporu. Bez nich by tyto práce nemohla vzniknout.

Obsah

Obsah ... 7
1 Úvod ... 8
2 Teorie vektorizace ... 9
2.1 Předzpracování obrazu ... 9
2.1.1 Jasové transformace ... 9
2.1.2 Prahování ... 10
2.1.3 Morfologické transformace ... 11
2.2 Segmentace ... 13
2.3 Detekce hran ... 14
2.3.1 Definice hrany ... 14
2.3.2 Hranové detektory ... 15
2.3.3 Cannyho detektor ... 16
2.4 Reprezentace hran ... 16
2.5 Postupy vektorizace ... 17
2.5.1 Vektorizace leteckých snímků ... 18
2.5.2 Houghova transformace ... 19
3 Návrh aplikace ... 20
3.1 Úvod ... 20
3.2 Vývojové prostředí ... 20
3.3 Vektorizační algoritmus ... 21
3.3.1 Princip algoritmu ... 21
4 Implementace ... 23
4.1 Knihovna OpenCV ... 23
4.1.1 OCV datové typy a makra ... 23
4.1.2 OCV funkce ... 23
4.2 Vlastní implementace ... 24
4.2.1 Uživatelské datové typy ... 24
4.2.2 Výkonné funkce a main() ... 24
4.3 Testování ... 25
5 Závěr ... 27
Literatura ... 28
Seznam příloh ... 29

1 Úvod

Vektorizace jako postup převodu obrazu rastrového na vektorový je v současné době velmi důležitým, byť ne úplně masově rozšířeným způsobem zpracování obrazových dat. Značného využití nachází především v oblasti geografie, kdy se do vektorové reprezentace převádějí buď již existující „papírové“ mapy, nebo satelitní či letecké snímky krajiny a vytvářejí se mapy digitální, další oblastí využívající vektorizaci je pak např. reklamní grafika, v níž často vyvstává potřeba plynule a neomezeně měnit velikost určitého obrazového objektu. Potřeba vektorizace je tedy řízena několika příčinami – získat reprezentaci zobrazovaných objektů nezávislou na rozlišení zobrazovacího zařízení, zmenšit datový objem ukládaného obrazu, který by byl v případě podrobných map neúnosný, a v neposlední řadě získat možnost vytvářet identifikované (a vektorově popsané) objekty v obraze. Další motivací k použití vektorizace může být i potřeba rozpoznávání takových objektů v obraze, především v rychle se rozvíjející oblasti počítačového vidění, kde se pak z rozpoznaných tvarů mohou sestavovat podoby těles v prostoru.

Ačkoliv vektorizace přináší jasné výhody, jejímu širšímu využívání brání některé její vlastnosti. Zaprvé je to její značná algoritmická náročnost a specifičnost, která prakticky znemožňuje používat jeden vektorizační nástroj na širší škálu vstupních obrazů. Odlišné požadavky jsou pochopitelně kladeny na zpracování naskenovaného vzoru fontu a satelitního snímku. K tomu se váže druhá nectnost, a to provázanost vektorizačního nástroje a výsledku – stejný obraz bude různými aplikacemi zpracován do různých podob a formátů. Nejzásadnější nevýhodu vektorové reprezentace obrazu bych však spatřoval v tom, že na rozdíl od rastrové, pro niž existuje nespočet datových formátů čitelných v libovolném prohlížeči na libovolném systému, vektorové reprezentaci tato univerzalita schází. Přestože existuje poměrně hojné množství formátů, které dokáží popisovat vektorová data, jsou zpravidla vázány na konkrétní aplikaci.

Tato bakalářská práce navazuje na přípravu ve formě semestrálního projektu z předcházejícího semestru, prakticky rozvíjí a testuje myšlenky v něm představené a snaží se prokázat jejich životaschopnost na demonstrační aplikaci.

V následující, druhé kapitole se tato práce věnuje teoretickým podkladům ke konverzi rastru na vektor, možnostem předzpracování vstupního obrazu pro separaci oblastí a detekci hran a způsobům nahlížení na reprezentaci hrany jako objektu zájmu v obrazu.

Kapitola třetí nastiňuje postup při návrhu konkrétního algoritmu a krátce zmiňuje vývojové prostředí OpenCV.

V kapitole čtvrté je popsána implementace demonstračního vektorizačního programu, který je programovou částí této práce. Je zde popsáno fungování programu, jednotlivé funkce a stručně také použité funkce knihovny OpenCV.

2 Teorie vektorizace

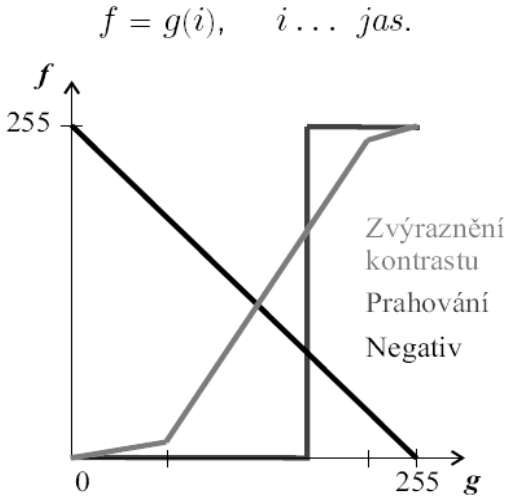
Vektorizace je poměrně komplexní úloha, která ve většině případů vyžaduje buď řízení ze strany uživatele, nebo pokročilou logiku v řízení automatickém. Důvodem jsou mimo jiné nejrůznější možnosti interpretace obrazových informací, které se snažíme ujednotit pomocí předzpracování obrazu. To by měly být úpravy ideálně takové, abychom odstranili z obrazu informace nežádoucí a zvýraznili informace podstatné. Informace žádoucí bývají v našem případě zpravidla hrany, které je možné v dalších fázích procesu nahradit pro vektorovou reprezentaci úsečkami a křivkami. Ke zvýraznění hran by tedy měly směřovat i zmiňované úpravy.

2.1 Předzpracování obrazu

Předzpracováním obrazu se rozumí jeho úprava pomocí jednodušších obrazových operací do podoby, která je následně vhodnější k složitějším operacím, vedoucím k vektorizaci. Těmito úpravami zpravidla nezískáváme žádné nové informace, naopak velmi často odstraňujeme informace nežádoucí, které může představovat šum, či oblasti obrazu, které nenesou z hlediska převodu obrazu na vektorový podstatné informace.

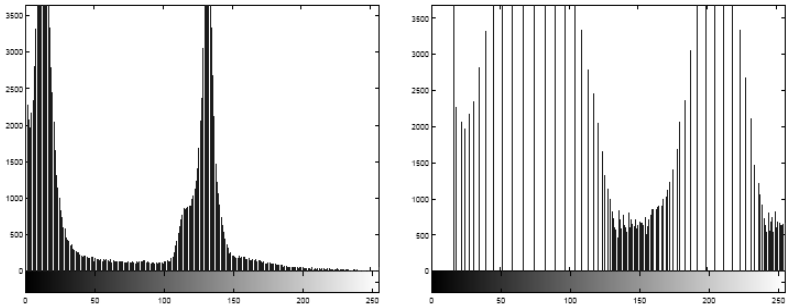
2.1.1 Jasové transformace

Jasové transformace[1] představují jedny z nejprimitivnějších úpravu obrazu, neboť se ve většině svých variant týkají pouze jednotlivých obrazových bodů (pixelů), případně jejich lokálního okolí. Příkladem úpravy obrazu ovlivňující pixely nezávisle je transformace jasové stupnice. Tato úprava bývá pro svou značnou jednoduchost často realizována hardwarově v zobrazovací kartě nebo monitoru počítače (pakliže není nutné provést ji natrvalo).



Obrázek 2.1 - Transformace jasové stupnice

Další transformací je ekvalizace histogramu. Vstupem pro tuto úpravu histogram obrazu *H(p)* s jasovou stupnicí $p=<p_0,p_k>$, a cílem je nalezení monotónní transformace jasové stupnice $q=T(p)$ takové, aby výsledný histogram *G(q)* byl rovnoměrný pro celý výstupní interval $q=<q_0,q_k>$. Důsledkem této transformace je pak zvýšení kontrastu a jasová normalizace obrazu, která je výhodná pokud se jedná například o fotografie reálného světa pořizované za různých světelných podmínek, které chceme automaticky porovnávat.



Obrázek 2.2 – Ekvalizace histogramu

2.1.2 Prahování

Prahování je velmi hojně užívaná jasová transformace, kterou lze snadno modifikovat pro aktuální potřeby. Je to operace, během níž se obraz převádí na binární, tedy na takový obraz, v němž se vyskytují pouze dvě hodnoty obrazového bodu. Toto je velmi výhodné, když se výstupní obraz chystáme v následujícím kroku strojově zpracovávat, neboť obraz se tímto rozdělí do vzájemně se nepřekrývajících podoblastí, neboli segmentuje. Rozhodování, zda bude pixelu přiřazena hodnota „0“, nebo „1“ se provádí porovnáním jeho hodnoty s tzv. prahem. Hodnota tohoto prahu (případně prahů) a způsob porovnávání se pak nastavuje podle druhu prahování.

Při jednoduchém prahování je hodnota prahu fixní a všechny pixely s hodnou menší než práh jsou nastaveny na „0“, pixely s hodnotou větší jsou nastaveny na „1“.

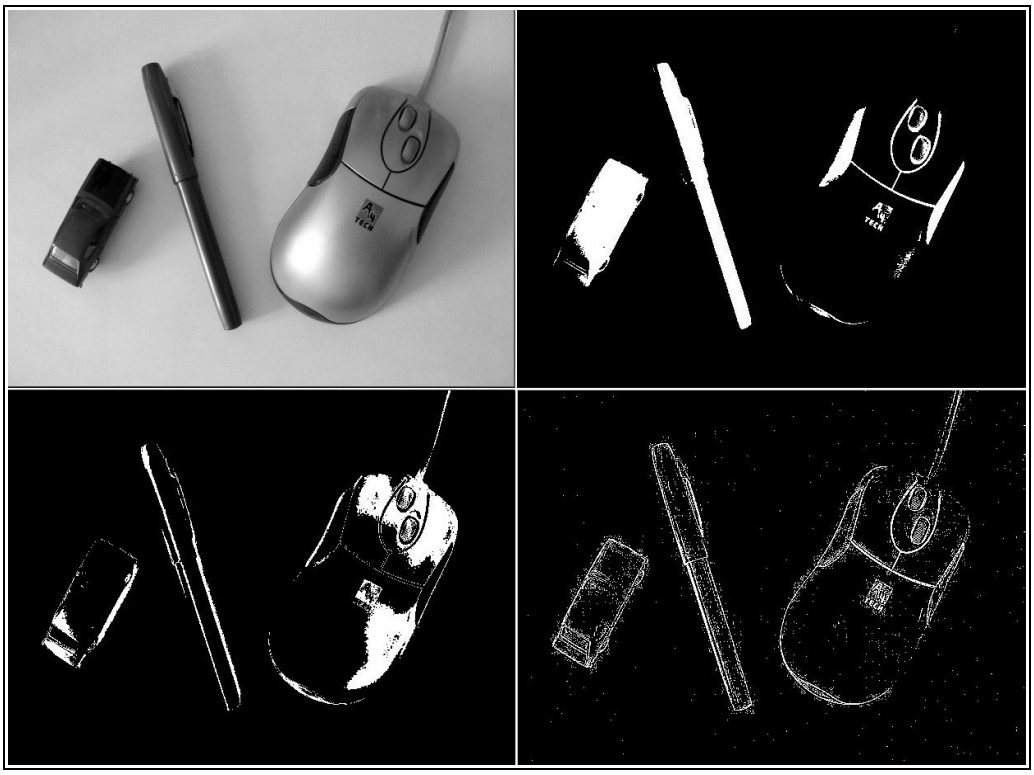
$$new(x,y)=(old(x,y)>threshold\ ?\ MAX : MIN)$$

Dvojité prahování používá rozmezí definované dvěma prahy, v němž se musí hodnota pixelu nacházet, aby získal hodnotu „1“.

$$new(x,y)=(threshold1>old(x,y)>threshold2\ ?\ MAX : MIN)$$

Problémem je pochopitelně stanovení hodnoty prahu tak, abychom z obrazu získali co nejvíce žádoucích informací. Proto se používají metody, kdy se tato hodnota nastavuje relativně vzhledem k hodnotám pixelů právě zkoumaného okolí. Při tzv. adaptivním prahování se hodnota prahu nastavuje v závislosti na hodnotách okolních bodů aktuálního pixelu, vyjádřených lokálním histogramem.

Následující obrázek demonstruje efekt jednoduchého prahování s prahem 64, dvojitého prahování s intervalem (64,96) a adaptivního prahování v obraze s 256-ti úrovněmi šedi:



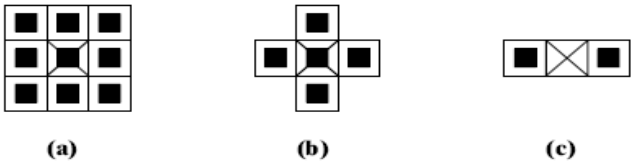
Obrázek 2.3 – Demonstrace prahování

Používají se i další metody prahování – mimo jiné třeba poloprahování, kdy pixely nad (či pod) určitou hodnotu jsou nastaveny na „0“ (nebo „MAX“), nebo oříznutí prahem, kdy se pixely s hodnotou pod (nad) nastaví právě na tuto hodnotu. Metodu prahování lze také spojit s důkladnější analýzou obrazu, tyto postupy jsou pak většinou vytvářeny „na míru“ aktuální potřebě následujícího kroku zpracování. Příkladem takové metody je prahování s hysterezí užívané funkcí cvCanny() z prostředí OpenCV (které bude podrobněji zmíněno níže), kdy jsou pixely s hodnotou vyšším než *threshold2* (vyšší práh) nastaveny na „1“, pixely s hodnotou nižší než *threshold1* (nižší práh) jsou nastaveny na „0“ a pixely s hodnotou v intervalu *threshold1<val(x,y)<threshold2* jsou nastaveny na „1“ pouze v případě, že přímo sousedí s již uznaným pixelem. Tímto postupem je možné redukovat zkreslení při detekci objektů například při neostrosti vstupního obrazu.

2.1.3 Morfologické transformace

V názvosloví digitálního zpracování obrazů rozumíme morfologií matematický nástroj, s jehož pomocí lze provádět jak předzpracování, tak i samotnou segmentaci obrazu.

Morfologické transformace Ψ [2] je dána relací mezi obrazem jakožto bodovou množinou X a typicky menší bodovou množinou tzv. strukturním elementem B. Aplikace morfologické transformace Ψ (X) se pak provádí systematickým posunem B po obraze X.



Obrázek 2.4 – Strukturní elementy B

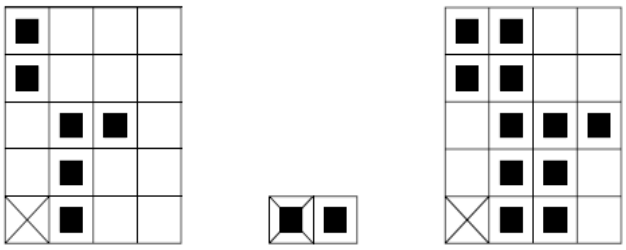
Ve vstupním obraze se bohužel nezřídka setkáváme se šumem, představovaným nežádoucími pixely či skupinami pixelů, které nemají návaznost na zobrazované objekty a nenesou obrazovou informaci. Pokud se od objektů dostatečně neodlišují svou jasovou složkou, je možné je odstraňovat na základě jejich velikosti a samostatnosti.Zde je možné s výhodou využít binární matematické morfologie. Ta pracuje s binárními obrázky, má definiční obor Z^2, obor hodnot $\{0,1\}$ a dvě základní operace, dilataci a erozi.

Dilatace je operace, při níž se vytváří tzv. Minkovského součet nad dvěma bodovými množinami, což může být vyjádřeno jako sjednocení posunutých bodových množin obrazu a strukturních elementů:

$$X \oplus B = \left\{ p \in \mathbb{E}^2 : \; p = x + b, \;\; x \in X \text{ and } b \in B \right\}$$

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

Dilatace se používá k zaplnění děr a malých průlivů v objektech, přičemž aplikací této operace se zvětšuje velikost objektu. Pakliže chceme, aby zůstala velikost objektu zachována, je nutné aplikovat ještě operaci eroze.

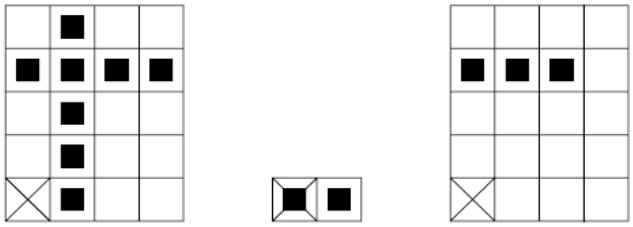


Obrázek 2.5 – Dilatace

Eroze je duální morfologickou transformací k dilataci, a jedná se o skládání dvou množin pomocí Minkovského rozdílu. Může být vyjádřena jako průnik všech posunů obrazu X o vektory $-b \in B$.

$$X \ominus B = \{ p \in \mathbb{E}^2 : \; p + b \in X \text{ pro každé } b \in B \} \qquad X \ominus B = \bigcap_{b \in B} X_{-b} \, .$$

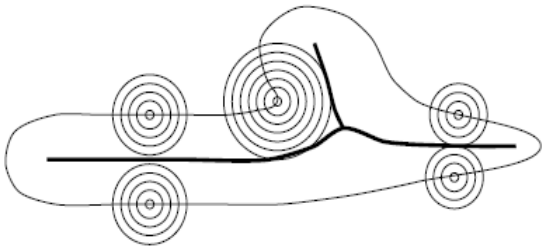
Pro každý bod obrazu *p* se ověřuje, jestli pro všechna možná *p+b* leží výsledek v *X*. Pokud ano, je výsledek 1, jinak 0. Použitím eroze vymizí objekty menší než strukturní element, například volné pixely nebo čáry tloušťky 1. Binární eroze se používá ke zjednodušení struktury.



Obrázek 2.6 - Eroze

Kombinací těchto dvou transformací získáme tzv. binární otevření a binární uzavření. Binární otevření je eroze následovaná dilatací, uzavření je dilatace následovaná erozí. Pokud se obraz *X* provedením binárního otevření (uzavření) strukturním elementem *B* nezmění, říkáme, že je tento otevřený (uzavřený) vzhledem k *B*. Otevření a uzavření jsou transformace idempotentní, neboli po jejich provedení se obraz jejich dalším opakováním již nezmění.

Sekvenční opakování binární eroze s vhodným strukturním elementem se používá při skeletonizaci, kdy je tento způsob hledání kostry objektu oproti vpisování kruhů (dle definice skeletu) jednak výpočetně lépe realizovatelný, jednak robustnější z hlediska odolnosti proti šumu.



Obrázek 2.7 – Kostra objektu

2.2 Segmentace

Segmentací se rozumí proces, při němž je obraz rozdělen do vzájemně se nepřekrývajících oblastí. Provádíme ji zejména tehdy, kdy další zpracování obrazu vyžaduje jasné oddělení těchto oblastí a homogenitu jejich vnitřních ploch. Techniky segmentace jako způsoby dosažení těchto vlastností obrazu mohou být různé, obecně je možné říct, že vybíráme vždy takovou metodu, aby její výstup úplně splňoval podmínky vstupu následující úpravy, a přitom aby zachovával co nejvíce informací z původního obrazu.

Segmentace může být provedena převodem na binární obraz (např. variantou prahování), pak získáme právě dva druhy oblastí, často se používá převod na víceúrovňový obraz, kdy každá úroveň představuje jeden druh oblasti. Příkladem algoritmu, který takový víceúrovňový obraz zpracovává je

níže zmíněný algoritmus pro vektorizaci leteckých snímků. Důvod, proč je segmentace zmiňována jako samostatná část procesu zpracování obrazu, je že pokud je příslušnost obrazového pixel do určité oblasti dána složitějším principem než např. pouhé jasovou úrovní (kdy je možné spolehnout se na primitivní metody typu prahování), stává se z rozdělení obrazu do oblastí netriviální operace, která si může vyžádat použitích značně komplexních algoritmů.

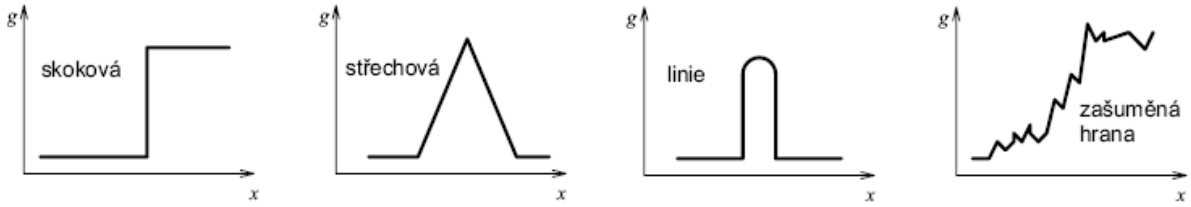
2.3 Detekce hran

Řadou výzkumů bylo prokázáno, že při vnímání vyšších organismů hrají nejvýznamnější roli v získávání informací z obrazu právě hrany, představované místy, kde se náhle mění hodnota jasu. Proto pokud chceme převádět rastrový obraz na vektorový, budeme se zajímat především o tyto oblasti, které pak budeme popisovat body a jejich propojeními.

2.3.1 Definice hrany

Hrana v obraze [3] popisuje rychlost změny a směr maxima růstu obrazové funkce *f(x,y)*, přičemž je vhodnou diskrétní aproximací tohoto gradientu. Hranový bod neboli edgel (od anglického edge element, odvozeno od pixel čili picture element) je potom takovým bodem v obraze, který se vyznačuje velkým modulem gradientu obrazové funkce.

Změna gradientu v okolí hrany může probíhat různým způsobem z hlediska rychlosti a průběhu této změny, jak ukazuje následující obrázek.



Obrázek 2.7 – Hrany

První tři případy představují jasové profily idealizovaných hran, čtvrtý graf demonstruje jasový profil hrany ve skutečném obrázku. Toto samozřejmě platí pro obecné obrazy, pokud provedeme např. segmentaci obrazu prahováním a převodem na binární obraz, můžeme dosáhnout profilu z prvního grafu, otázkou však je, jak přesně kopírují takto získané hrany ty skutečné.

2.3.2 Hranové detektory

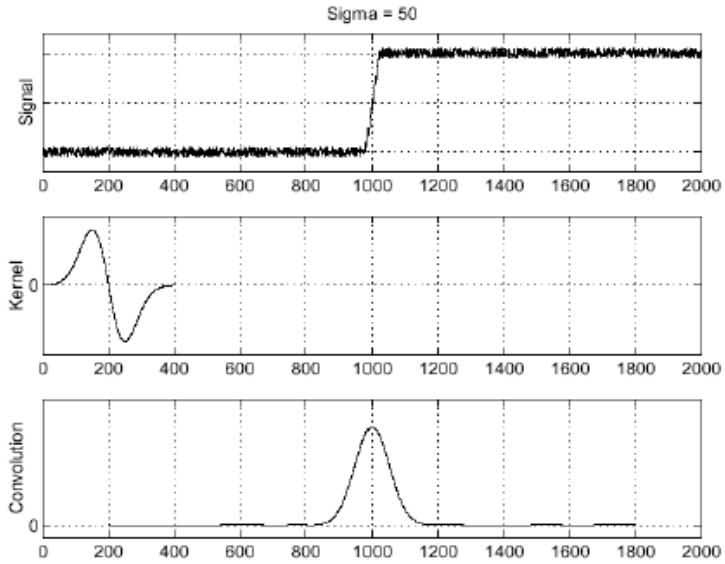
Pro zjišťování hran v obraze je možné použít celou řadu tzv. hranových detektorů. Ty bývají založeny na následujících principech:

1. hledání maxim prvních derivací (Roberts, Prewittová, Sobel, Canny)
2. hledání průchodů druhých derivací nulou (Marr-Hildreth)
3. lokální aproximace obrazové funkce parametrickým modelem, např. polynomem dvou proměnných (Haralick)

Zatímco gradient jednorozměrného signálu je roven jeho derivaci, u obrazu jakožto dvojrozměrného signálu je jeho výpočet složitější. Gradient spojité funkce *n* proměnných je roven vektoru parciálních derivací, což pro jeho velikost a směr u dvourozměrného signálu představuje následující vztah:

$$\|\nabla f(x,y)\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}, \quad (\sin\psi, \cos\psi) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$$

V diskrétním prostředí je třeba derivace vhodně aproximovat, a to buď rekonstruováním spojité funkce a následným spočtením derivace, nebo aproximací derivace konečnými diferencemi. Problém s výpočtem derivací nastává, pokud je signál znečištěn šumem. Lokální maxima a minima v signálu pak prakticky znemožňují nalezení „té správné“ změny v obraze. Tento problém je možné řešit konvolucí vstupního (zašuměného) signálu vhodnou konvoluční maskou, která průběh funkce vyhladí a derivací pak nalezneme již jen jedno maximum. Vzhledem k asociativitě konvoluce a derivace je možné tyto možné shrnout do jediného operátoru.



Obrázek 2.8 – Konvoluce s derivací

Tyto operátory (konvoluční masky) jsou známé většinou pod jmény jejich tvůrců a využívají tzv. 8-okolí pixelu, tedy masku 3x3 body s aktuálním pixelem uprostřed. Jejich efekt na zvýraznění hran je různý, záleží i na způsobu jejich použití, asi nejvhodnějším kandidátem na post „nejlepšího“ způsobu získávání hran je však Cannyho hranový detektor.

2.3.3 Cannyho detektor

Cannyho detektor využívá většina aplikací provádějících detekci hran a jsou k němu volně dostupné implementace. Jeho algoritmus je zjednodušeně následující:

1. Najdi přibližně směry gradientu
2. Pro každý pixel najdi jednorozměrnou derivaci ve směru gradientu pomocí vhodné masky
3. Nalezni lokální maxima těchto derivací
4. Získej hranové body prahováním s hysterezí
5. (Proveď syntézu hran získaných při různých úrovních vyhlazení)

Poslední bod se většinou pro značné navýšení výpočetní náročnosti neprovádí. Prahování s hysterezí bylo zmíněno již výše, v tomto algoritmu má ten účel, aby byly spolehlivě a bez mezer nadetekovány výraznější hrany a potlačeny menší, nevýrazné a zpravidla méně významné. Edgel je tedy zachován, pouze pokud je součástí výraznějšího řetězce hrany, a hrany nedosahující žádným bodem hodnoty horního prahu jsou eliminovány.

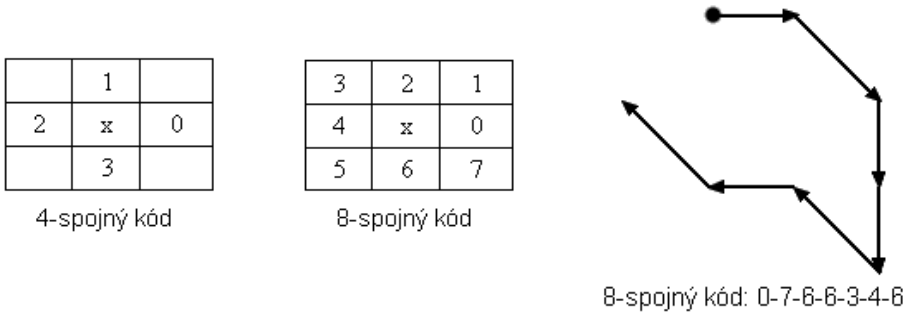
Problémem při detekci hran však zůstává skutečnost, že požadavky na její schopnosti jsou odpočátku do jisté míry protichůdné. Chceme totiž, aby se spolehlivě a s dobrou lokalizací zjistila pokud možno každá existující hrana a přitom aby nebyly nadetekovány pokud možno žádné neexistující hrany. Ale čím lepší je detekce, tím horší se stává lokalizace, popř. tím více neexistujících hran se detekuje. Toto dilema se promítá do volby měřítka vyhlazení (nejen) u Canny detektoru – zvolíme-li větší měřítko, získáme větší odolnost proti šumu, ale ztrácíme více nevýrazných hran a snižuje se přesnost lokalizace hran.

2.4 Reprezentace hran

Hranu můžeme zaznamenat v binárním obraze nastavením patřičných pixelů, tato forma však není příliš paměťově efektivní, ani úplně ideální z hlediska zkoumání vlastností hrany za účelem vektorizace. Lepší možnosti nabízí uložení hrany v tzv. řetězovém kódu (chain code) [4]. Ten zaznamenává pouze vzájemnou polohu dvou sousedních hranových bodů, čímž výrazně redukuje datový objem a zároveň poskytuje vhodnou strukturu pro některé druhy operací.

Tento způsob ukládání hran pochází z hlavy amerického inženýra Herberta Freemana, proto se řetězové kódy někdy nazývají Freemanovy (Freeman chain codes). Ten je vytvořil jako odpověď na potřebu najít způsob, jak by počítač mohl porozumět čárovým kresbám.

Řetězový kód ukládá linie pomocí určení směru, v němž se vzhledem k aktuálnímu bodu nalézá bod následující. Pokud jsou tyto směry čtyři (a pracujeme s čtyřokolím bodů), nazýváme kód čtyřspojný (four-connected), pokud je směrů osm, mluvíme o osmispojném kódu (eight-connected). Jestliže používáme osmispojný kód, stačí nám k uložení každého dalšího bodu řetězu teoreticky jen tři bity, pokud bychom použili čtyřspojný, byly by to pouze dva bity, což je myslím dostatečným důkazem efektivity uložení takto reprezentované linie.



Obrázek 2.9 – Řetězové kódy

Další výhodou této reprezentace hran je fakt, že je relativní, tudíž invariantní vůči rotaci a posunu. Pokud chceme hranu např. z důvodu zkoumání jejího tvaru a pro zjednodušení početních operací posunout, stačí posunout její počáteční bod, v případě potřeby jejího otočení stačí každému prvku řetězu na třech bitech přičíst nebo odečíst patřičnou hodnotu. Této skutečnosti je využito při zpracovávání hran v algoritmu, který je implementován v programové části této práce a je popsán výše.

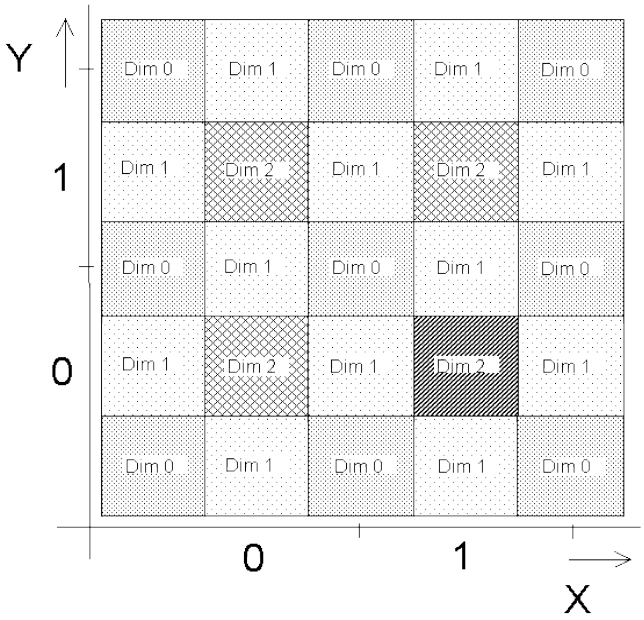
2.5 Postupy vektorizace

Komerčně dostupných aplikací pro převod rastru na vektor je poměrně slušná řada a poslední dobou se objevují i jejich volně dostupné alternativy, konkrétní vektorizační algoritmy jsou však stále předmětem firemního know-how a nebývají volně publikovány. s trochou štěstí lze ještě dohledat na internetu studentské práce k tomuto tématu, tady se zase však většinou jedná o aplikace úzce zaměřené například na hledání konkrétního tvaru v obraze (např. v medicíně – fotografie defektů, zranění atd.), což má s vektorizací společnou jen určitou část postupu.

2.5.1 Vektorizace leteckých snímků

Přesto existují vyjímky - jeden z postupů, konkrétně orientující se na vektorizaci leteckých snímků, může představovat následující algoritmus [5]. Do vstupního obrazu, představovaného typicky leteckým snímkem, který byl již vhodnou metodou segmentován do víceúrovňového obrazu (viz výše), jsou uměle vloženy nové body na hrany a na rohy stávajících pixelů.

Takto vznikne nový obraz (již binární), v němž tyto pomocné body vyjadřují topologické vazeb mezi pixely, s nimiž sousedí. V novém obraze je pixel nastaven pouze v případě, že leží mezi odpovídajícími dvěma body v původním obraze, které mají rozdílnou hodnotu. Nově vzniklý binární obraz tedy zaznamenává již pouze vztahy mezi jednotlivými okrajovými pixely sousedících oblastí, neboli hrany.



Obrázek 2.10 – Algoritmus vložení bodů

v dalším průchodu pak algoritmus vyhledává uzlové body podle nenulových sousedů a následujícím průchodem (prováděným pro dokonalejší fungování dvakrát) tyto uzlové body vzájemně přiřazuje. Následuje další průchod, při němž se kontrolují plochy a uzavírají se do jednotlivých celků, vyřazují se plochy nesplňující rozměrová kritéria a kontroluje se, zda nebyla plocha zjištěna dvakrát. V posledním průchodu se budují topologické struktury.

Je nutno dodat, že ač popisovaný algoritmus velmi kvalitně převádí segmentovaný obraz do vektorové podoby, autor sám upozorňuje na značnou paměťovou náročnost tohoto postupu, během nějž se vytvářejí datové struktury o více než čtyřnásobném počtu prvků, než byl počet pixelů původního snímku.

2.5.2 Houghova transformace

Jisté možnosti k vektorizaci poskytuje také Houghova transformace. Ta se používá k detekci obrazových primitiv, jejichž tvar je vyjádřitelný anulovanou rovnicí, tedy $f(a_1, ..., a_m, x, y)=0$ a pracuje tak, že mapuje body či skupiny bodů ze vstupního vektorového prostoru na parametrický prostor definovaný touto rovnicí. Pokud se tak hledá např. kružnice, hodnotí se pro každý zkoumaný bod kandidáti na středy kružnic procházejících tímto bodem. Kandidátní body, které získají tímto způsobem hodnocení nad určitý limit jsou pak jako určeny jako středy kružnic.

Houghova transformace je velmi robustní metodou, která je odolná vůči šumu, a dokáže se vyrovnat i s chybějící částí detekovaného obrazového elementu. Konvertovat rastrový obraz na vektorový tímto způsobem je však evidentně krajně nepraktické vzhledem k faktu, že nejen musíme znát tvar a pokud možno i hrubé rozměry hledaného objektu, ale především ho musíme být schopni popsat rovnicí ve výše uvedeném tvaru. Proto se používá především pro lokalizaci očekávaných obrazových primitiv. Provádění Houghovy transformace je také značně výpočetně náročné, což může a nemusí být hledisko pro výběr metody vektorizace.

Vzhledem k návrhu vektorizačního algoritmu popsaného níže by bylo možné použít Houghovu transformaci pro linie, tady by však schopnost ignorovat vynechané úseky linie mohla být nežádoucí, neboť tyto „mezery“ mohou nést klíčové informace. Z těchto důvodů a také pro rozvedení a ověření mého původně navrhnutého algoritmu sledování rovných linií v řetězech hran jsem se rozhodl Houghovu transformaci nepoužít.

3 Návrh aplikace

3.1 Úvod

Vzhledem k nedostatku volně dostupných informací o vektorizačních algoritmech jsem při řešení tohoto projektu začínal prakticky od nuly. Rozhodl jsem se proto pro zkoumání postupů pro vektorizaci jednodušších obrazových objektů, s jakými se setkáváme například při zpracovávání návrhů fontů či převádění návrhů nejrůznějších firemních log do vhodnějšího (vektorového) tvaru.

Poněvadž zadání bakalářské práce podrobněji nespecifikovalo podrobnosti ohledně vyžadovaného vstupu a výstupu, popřípadě formě aplikace (či knihovny), rozhodl jsem se pro aplikaci spouštěnou z příkazové řádky, jíž bude vstupní obraz předán parametrem spuštění. Tato forma umožňuje jak určitou míru komunikace s uživatelem (jako demonstrační, popř. testovací aplikace), tak i snadnou úpravu na samostatně fungující proces, který může být přednastaven a spouštěn programově za účelem zpracovávání série obrazů. V současné podobě aplikace vektorová data neukládá, pouze je nashromáždí a před ukončením chodu pro kontrolu úspěšnosti převodu opět vykreslí. Implementovaný algoritmus je zaměřen na zajištění přímo jdoucích linií v obraze a na jejich reprezentaci počátečním a koncovým bodem, a jeho zamýšlená oblast použití je vektorizace vzorů fontů, symbolů a log.

3.2 Vývojové prostředí

Volba vývojového prostředí rozhodně nebyla podružnou částí přípravy projektu. Vycházel jsem pochopitelně ze znalosti programovacích jazyků, takže přestože například program Matlab poskytuje vhodné nástroje pro zpracování signálů (i obrazových), vzhledem k nedostatku zkušeností s ním jsem si jej použít příliš netroufal. Jako určitá možnost se jevilo použití mně bližšího jazyka C/C++ v kombinaci s knihovnou GLUT, se kterou jsme pracovali ve čtvrtém semestru ve cvičeních předmětu Základy počítačové grafiky.

Během hledání informací o postupech vektorizace a přípravy obrazů k tomuto procesu jsem narazil na knihovnu, která se ukázala jako mimořádně vhodná pro mé potřeby. Jedná se o knihovnu OpenCV [6] pro počítačové vidění, určenou pro jazyk C/C++ a volně dostupnou jako open source. Tato knihovna byla původně vyvinuta společností Intel, je nezávislá na platformě (pracuje jak pod operačním systémem Windows, tak pod Linuxem) a je optimalizovaná a zamýšlená pro použití při real-time zpracování obrazových informací. Obsahuje velké množství funkcí pro načítání a ukládání jak statických obrázků, tak videa, případně jejich získávání z periferních zařízení, nižších i vyšších API (Aplication Programming Interface – Aplikační programovací rozhraní) funkcí a poskytuje rozhraní pro technologii Integrated Performance Primitives, implementovanou na procesorech Intel.

Důvod proč jsem se rozhodl využít této knihovny je především ten, že poskytuje vynikající základnu pro práci s obrazem na úrovni jednotlivých bodů, a zároveň implementuje v podobě snadno použitelných funkcí de facto všechny úpravy pro předzpracování obrazu, kterých jsem zamýšlel využít (či testovat jejich užití), tak jak jsou popisovány v předchozí kapitole. Díky integrovaným API funkcím se také velmi usnadňuje řízení aplikace pomocí klávesnice či vytváření oken a zobrazování obrázků k demonstraci či kontrole průběhu.

Knihovnu OpenCV je možné používat v kombinaci s prakticky libovolným překladačem C/C++ či přímo vývojovým prostředím. Zvolil jsem si tedy prostředí Microsoft Visual Studio 2005, na nějž studentům FIT poskytuje bezplatnou licenci pro školní využití, a to vzhledem k pohodlnosti tvorby programů a jejich ladění v něm. Svou roli sehrály i předchozí zkušenosti s tímto prostředím a především možnost snadného zahrnutí knihovny OpenCV do vyvíjeného projektu.

3.3 Vektorizační algoritmus

Součástí semestrálního projektu minulý semestr bylo navrhnout algoritmy pro vektorizaci rastrového obrazu. Při tvorbě algoritmu pro vektorizaci jsem tedy stavěl na těchto základech.

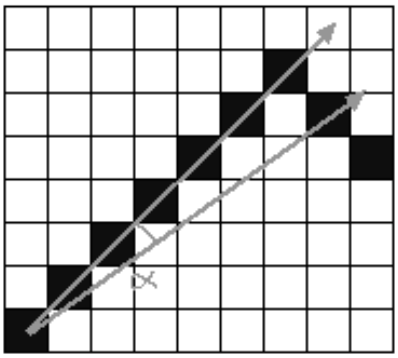
První navrhovaný algoritmus předpokládal nalezení uzlových bodů v obraze a následné sledování hran a dohledávání propojení mezi nimi. Obě tyto operace jsou však netriviální a jak naznačuje implementace algoritmu založeného na podobném principu, zmiňovaného v kapitole 2.5.1, může být i výkonově a paměťově značně náročná. Druhý návrh předpokládal kvalitní detekci hran (kterou je možné díky nástrojům prostředí OpenCV do značné míry zaručit), po níž mělo následovat sledování průběhu hrany a odhadování, zda edgely (hranové body) v aktuálně sledované části reprezentují projekci do rastru nějaké polopřímky. Tento postup mi připadal pro určité účely použitelný, a proto jsem se jej rozhodl dále rozvést.

3.3.1 Princip algoritmu

Nejdříve je třeba vstupní obrazu vhodně předpřipravit a provést kvalitní detekci hran, pro což se ukázal být vhodný Cannyho hranový detektor. Následuje sekvenční procházení takto upraveného obrazu s cílem nalézt nenulový bod, jehož poloha je pak uložena a je od něj zahájeno prohledávání a ukládání hrany, jejímž je členem. Tato hrana je ukládána do osmispojného řetězového kódu, přičemž se preferují kroky do čtyřokolí před kroky do diagonálních sousedních bodů, aby se předešlo vynechání bodu či jinému nekonzistentnímu chování v místě, kde více pixelů sousedí oběma způsoby. Během ukládání hran (kontur) do řetězů jsou jednou zajištěné pixely vynulovány, aby se zabránilo jejich vícenásobnému zpracování.

Jakmile jsou všechny kontury vstupního obrazu načteny, uloženy v řetězových kódech a všechny body vstupního obrazu vynulovány, začíná zpracování takto získaných dat. Princip vektorizačního algoritmu spočívá v procházení hranových pixelů a hledání úseků o určité minimální

délce, které si udržují stabilní směr podle určitého kritéria tolerance. Jakmile je tato tolerance překročena, je buď dosud zkoumaný úsek uznán za úsečku (překročí-li minimální délku) a uložen, nebo je posouzen jako součást křivého úseku a ponechán beze změny, přičemž se pokračuje ve vyhledávání rovných úseků od bodu těsně před onou odchylkou, která způsobila předchozí ukončení.



Obrázek 3.1 – Odchylka od směru

Rozhodování o přímosti prozkoumávaného úseku je založeno na porovnávání úhlu, definovaného vzájemnou polohou nově prozkoumávaného článku řetězu a jeho počátkem, s úhlem určeným dosavadním průběhem zkoumaného úseku řetězu. Pokud je tento v toleranci odvozené od vzdálenosti počátečního a zkoumaného bodu, je dosavadní průběh zhodnocen jako přímý úsek a pokračuje se porovnáváním úhlů pro další bod.

Jelikož je řetězový kód (kap. 2.4) invariantní vůči otočení a posunu, mohu si hranu jím reprezentovanou určitou korekcí hodnot článků řetězu a posunutím počátečního bodu transformovat do vhodné polohy a vhodného (bezpečného) intervalu pro počítání funkce arcus tangens. Podrobněji je tento postup rozebrán v implementační části.

4 Implementace

V této části práce se nachází popis struktury demonstrační aplikace, a také popis vektorizačního algoritmu včetně jeho implementačních detailů.

4.1 Knihovna OpenCV

Knihovna OpenCV implementuje celou řadu funkcí, které jsou použitelné jak pro práci s obrazem, tak pro grafickou i textovou komunikaci s uživatelem.

Následuje výčet funkcí a datových typů knihovny OpenCV (dále jen OCV), které jsem využil při vytváření demonstrační aplikace a jejich stručný neformální popis.

4.1.1 OCV datové typy a makra

- *IplImage** - typ ukazatel na strukturu, která po načtení obrazu obsahuje informace o něm i jeho data; používá se jako odkaz na obrázek ve všech OCV funkcích, které jej vyžadují, např. vykreslení obrázku, operace nad obrázkem atd. V aplikaci využívám položky struktury:
  - width – šířka obrazu
  - height – výška obrazu
  - imageData – jednorozměrné pole obrazových dat
  - widthStep – krok pro posunutí o *y+-1*, pixel *x,y* se adresuje *data[x+step*y]*
- *CvPoint* – typ struktura, pro uložení pozice bodu, obsahuje položky *.x* pro určení pořadí bodu na řádku a *.y* pro určení řádku, na němž se nachází.
- *CvScalar* – typ struktura, má jedinou položku *.val*, která se užívá k uložení barevné informace.
- *CV_RGB(r, g, b)* – makro pro přetypování tří přímých hodnot na typ CvScalar
- *CV_IMAGE_ELEM(img, uchar, i, j)* – makro používané pro přístup na obrazový bod, předává se odkaz na *IplImage* strukturu, datový typ pro přetypování a *i,j* poloha bodu

4.1.2 OCV funkce

- *IplImage* cvLoadImage(filename, iscolor);* - funkce pro načtení obrázku ze souboru identifikovaného parametrem *filename,* podle hodnoty *iscolor* se barevná informace buď ponechá původní, nebo se vynutí barva či stupně šedi.
- *void cvCanny(image, edges, threshold1, threshold2, aperture=3);* - tato funkce implementuje detekci hran pomocí Cannyho hranového detektoru (kap. 2.3.3). Načítá obraz *image*, výsledek pak ukládá do *edges*. Provádí prahování s hysterezí s prahy určenými parametry *threshold1,* threshold2, nepovinný parametr *aperture* je pro Sobelův operátor použitý v rámci algoritmu Canny.

- *int cvNamedWindow(name, flag=1);* - funkce pro vytvoření pojmenovaného okna (jménem *name*), flag udává příznak otevření okna (1 znamená automatická velikost).
- *void cvMoveWindow(name, x, y);* - funkce pro nastavení polohy okna idetifikovaného parametrem *name*, vzdálenost od levého horního rohu určují *x,y*.
- *void cvShowImage(name, img);* - funkce, která zobrazí obrázek identifikovaný *img* do okna jménem *name*.
- *int cvWaitKey(delay=0);* - tato funkce očekává stisknutí klávesy, a pak vrací její ASCII kód. Volitelný parametr *delay* určuje maximální dobu čekání v milisekundách, pokud je 0, čeká se neomezeně dlouho.
- *CvPoint cvPoint(x, y);* - funkce, vracející strukturu *CvPoint* naplněnou parametry *x, y*.
- *void cvReleaseImage(img);* - funkce pro uvolnění obrázku *img*.

4.2 Vlastní implementace

Následuje stručný popis implementace demonstrační aplikace, kde jsou popsány nově vytvořené datové typy pro ukládání a práci se získávanými obrazovými informacemi a kde jsou stručně popsány výkonné funkce volané v aplikaci a tělo aplikace main().

4.2.1 Uživatelské datové typy

Pro ukládání kontur a nadetekovaných linií používám STL kontejnery *vector* vytvářené z uživatelských typů. Používané nové typy jsou tedy následující:

- *std::vector<uchar> vec_uch;* - vektor k uložení samotného řetězového kódu. Přestože by se jeden prvek řetězového kódu dal teoreticky uložit na třech bitech, pro zjednodušení implementace je použit typ *unsigned char.*
- *struct {CvPoint start; vec_uch chain;} t_contour;* - struktura obsahující jednu konturu,jejími položkami jsou počáteční bod *start* a vektor *vec_uch chain* pro řetězový kód.
- *std::vector<t_contour> vec_contours;* - kontejner slouží k uložení všech kontur *t_contour,* které byly načteny.
- *struct {CvPoint start; CvPoint end;} t_line;* - struktura ze dvou bodů *CvPoint*, slouží k uložení nalezené úsečky.
- *std::vector<t_line> vec_lines;* - vektor nalezených úseček tvořený strukturami *t_line*.

4.2.2 Výkonné funkce a main()

Tato sekce obsahuje stručný popis fungování výkonných funkcí programu, detailní průchod kódem těchto funkcí je v dokumentu Implement_details.pdf na přiloženém CD.

- *int find_lines(vec_contours * conts, vec_lines * lines, int * neibs, IplImage * img, const char min_l, const float tolr);* - Tato funkce slouží k prohledání vektoru uložených kontur *conts*. Hlavním cyklem funkce prochází všechny prvky tohoto vektoru, a pro každý z nich zavolá funkci *detect_line();*, která jí vrátí hodnotu větší nuly, pokud nalezne v předané kontuře řetěz odpovídající rovnému úseku. Tento rovný úsek je pak uložen jako dvojice bodů do vektoru *lines*, zbývající část kontury je uložena k dalšímu hledání.

- *int detect_line(t_contour act_cont, int s_i, int e_i, t_line line, const char min_l, const float tolr);* - Tato funkce dostane v parametrech řetězový kód kontury ve vektoru *chain* a odkazy na proměnné *s_i a e_i*, které budou v případě nalezení rovného úseku nastaveny způsobem popsaným výše. Řetěz *chain* si pro lepší manipulaci transformuje a pak jej pro každý jeho prvek testuje, zda se neodchýlil od směru daného jeho předchozími body. Pokud se odchýlil a byla mezitím překročena minimální platná délka úsečky, je tato pomocí indexů vrácena; pokud nebylo dosaženo minimální žádané délky, provede se znova transformace řetězu a prohledává se od nynější pozice.

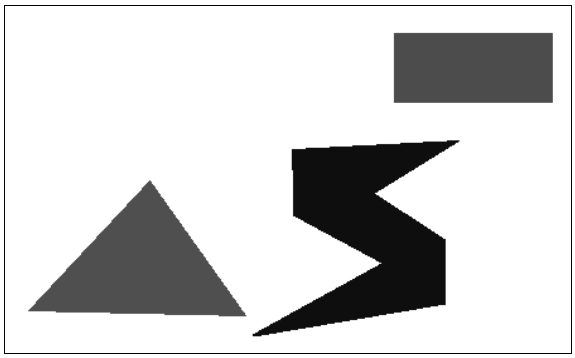
- *int in_tolerance(CvPoint p,double * sum_a,int * num_a, const float tol_ratio)*; - Rozhodovací funkce, která určí, zda se nový bod *point* svou polohou odchyluje od dosavadního průběhu linie.

- *int get_contours(vec_contours *conts,IplImage * img, int * neibs);* - Tato funkce prochází vstupní obraz *img* a hledá pixel s nenulovou hodnotou. Jakmile jej najde, začne od tohoto bodu vždy prohledávat okolní body (s preferencí čtyřokolí) a hledat souseda; když ho najde, posune se na jeho pozici, předešlý bod smaže a posun si uloží do řetězu. Až už nenajde žádného souseda, uloží získanou konturu do *conts* a pokračuje v prohledávání obrazu pro další kontury.

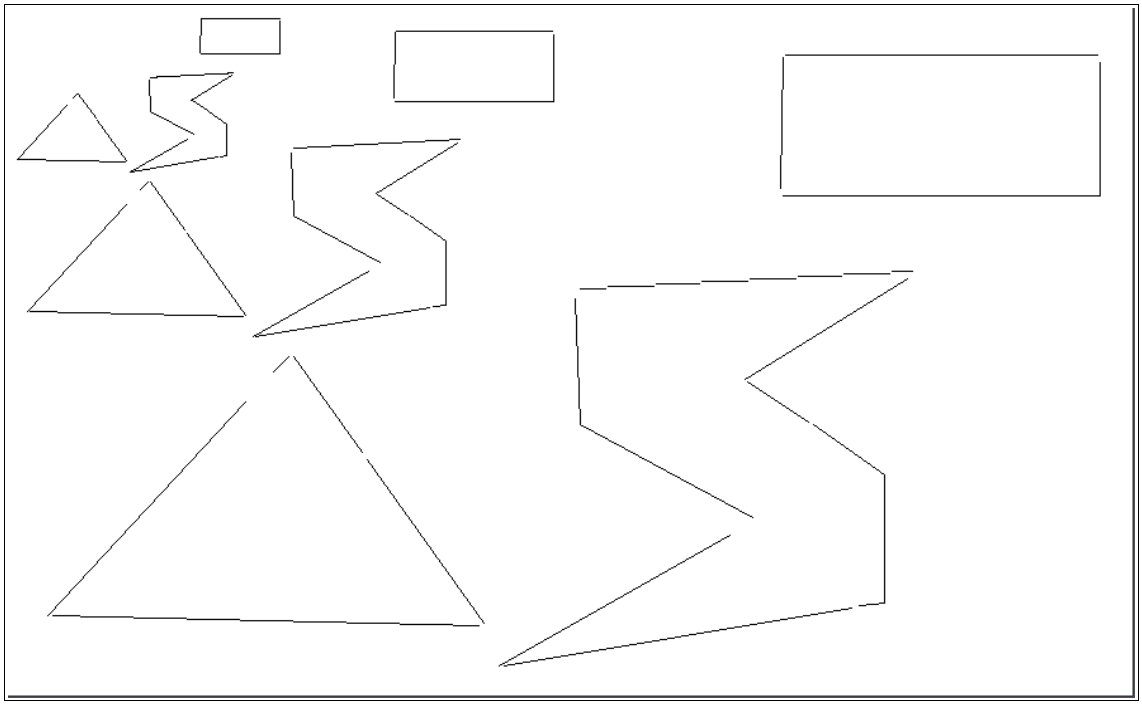
- *int Main(int argc, char *argv[]);* - Funkce main má na starosti pouze zpracovává parametrů spuštění aplikace a volání příslušných funkcí . Obstarává komunikaci s uživatelem vytvořením okna aplikace a zobrazováním dílčích výsledků, na určitých místech očekává stisky kláves a v daných chvílích volá patřičné výkonné funkce, což jsou v tomto pořadí *cvCanny(); , get_contours(); a find_lines();*.

4.3 Testování

Rané fáze testování odhalovaly množství chyb v prvotních podobách aplikace, které se však naštěstí dařilo odstraňovat bez razantních změn v řízení chodu programu. Z celkového pohledu bylo testování úspěšné, podařilo se mi zprovoznit aplikaci v celém jejím implementovaném rozsahu.



Obrázek 4.1 – Vstupní obraz



Obrázek 4.2 – Výstup

Pro představu o detekčních schopnostech vektorizačního algoritmu ve spojení s Cannyho detektorem uvádím příklad vstupního obrazu (obrázek 4.1) a kontrolního výstupu (obrázek 4.2, s dvojnásobným rozlišením), v němž jsou detekované linie vykresleny v původní velikosti, v polovičním a ve dvojnásobném měřítku.

Více výsledků testování je zaznamenáno ve složce "tests" na přiloženém CD. Pro ilustraci fungování algoritmu jsem obraz po vektorizaci opět vykreslil se stejným rozlišením.

5 Závěr

Během vývoje algoritmu a následně demonstrační aplikace jsem se potýkal s celou řadou problémů, počínaje nedostatkem teoretických podkladů a konče odstraňováním chyb z vyjímečných stavů v aplikaci. Vzhledem k těmto průtahům se mi nepodařilo rozvinout aplikaci do takové formy, aby byla prakticky použitelná, nicméně dosáhl jsem dílčího úspěchu potvrzení funkčnosti navrhovaného algoritmu. Další vývoj by zahrnoval nastudování používaných vektorových formátů a implementace některého z nich do aplikace tak, aby byly výsledky prezentovány dále zpracovatelnou formou, případně doplnění aplikace o nějakou vhodnou metodu zpracování kontur, které nebyly rozpoznány jako úsečky, například jejich interpolace pomocí Catmull-Rom splinů. O vytváření složitějšího grafického rozhraní neuvažuji, neboť aplikace byla koncipována tak, aby ji bylo možné řídit parametry předanými při spuštění, a je tedy určena pro programové spouštění. Současné grafické rozhraní má pouze demonstrační účely a do budoucna bych počítal s jeho odstraněním.

Při řešení projektu jsem se seznámil s celou řadou postupů pro úpravy vstupního obrazu, detekci hran v obraze a některými možnostmi pro hledání a rozpoznávání objektů v obraze. Blízce jsem se přesvědčil o netriviálnosti hledání vektorové reprezentace diskrétního dvojrozměrného signálu a navzdory prvotním potížím se mi podařilo vytvořit a implementovat relativně jednoduchý algoritmus pro identifikaci a uložení rovných úseků hran v obraze.

Literatura

[1] Hlaváč, V., *Předzpracování v prostoru obrazů* (http://cmp.felk.cvut.cz/~hlavac/Public/TeachingLectures/PredzpracObr.pdf).

[2] Hlaváč, V. *Matematická morfologie* (http://cmp.felk.cvut.cz/~hlavac/Public/TeachingLectures/BinMatMorfolCesky.pdf).

[3] Hlaváč, V. *Hledání hran* (http://cmp.felk.cvut.cz/~hlavac/Public/TeachingLectures/DetekceHran.pdf).

[4] Image processing fundamentals - Contour representation (http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip-Contour.html).

[5] Fojtík, J. *Algoritmy vektorizace leteckých snímků* (http://cmp.felk.cvut.cz/~fojtik/vectoris/vektoris.htm).

[7] Krpec, V. *Využití Houghovy transformace při segmentaci biomedicínských obrazů* (http://www.razdva.cz/vencik/dp/DP2003.pdf).

[6] Intel - Open Source Computer Vision Library (http://www.intel.com/technology/computing/opencv/index.htm).

Seznam příloh

Příloha 1. CD s kompletním Visual Studio projektem demonstrační aplikace, detaily implementace a vzorky výsledků testování aplikace. CD je vloženo v zadních deskách této práce.