VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

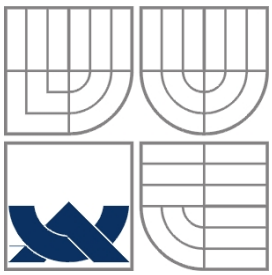
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PŘEVOD KŘIVKY Z RASTRU NA VEKTOROVOU REPRESENTACI

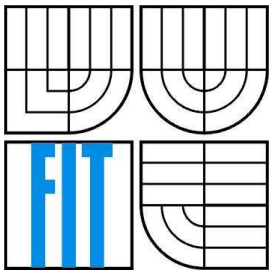
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE JIŘÍ KRÁL
AUTHOR

BRNO 2007



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

PŘEVOD KŘIVKY Z RASTRU NA VEKTOROVOU REPREZENTACI

CONVERSION OF RASTER-CURVE INTO VECTOR REPRESENTATION

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ KRÁL

VEDOUCÍ PRÁCE
SUPERVISOR

ING. VÍTĚZSLAV BERAN

BRNO 2007

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání bakalářské práce

Řešitel: **Král Jiří**
Obor: Informační technologie
Téma: **Převod křivky z rastru na vektorovou representaci**
Kategorie: Zpracování obrazu

Pokyny:

1. Prostudujte dostupnou literaturu týkající se tématu. Proveďte rozbor dosavadního stavu problematiky.
2. Na základě nastudovaných informací navrhněte postup řešení zadání. Diskutujte zvolené metody a vysvětlete navržené řešení.
3. Vytvořte anotovanou testovací sadu dat pro závěrečné testování Vašeho řešení.
4. Navrhněte a implementujte knihovnu realizující Vámi navržený postup. Dbejte na programovou čistotu, modulárnost a přenositelnost knihovny. Knihovnu dokumentujte.
5. Vyhodnoťte Vaše řešení pomocí testovacích dat, srovnejte Vaše výsledky s výsledky jiných prací a proveďte jejich diskuzi.

Literatura:

- Žára, J., Beneš, B., Felker, P.: Moderní počítačová grafika, Computer Press, 1998.
- Watt, A., Policarpo, F.,: The Computer Image, Addison Wesley.
- Watt, A.: 3D Coputer Graphics, Addison Wesley,1993.
- Rafael C. Gonzalez, Richard E. Woods: Digital image processing, Prentice-Hall, 2002.

Při obhajobě semestrální části projektu je požadováno:

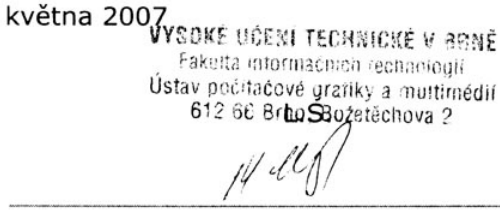
- Body 1., 2., 3. a částečně bod 4.

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: **Beran Vítězslav, Ing.**, UPGM FIT VUT
Datum zadání: 1. listopadu 2006
Datum odevzdání: 15. května 2007



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: **Jiří Král**
Id studenta: 88677
Bytem: Tyršova 3B, 612 00 Brno
Narozen: 04. 11. 1983, Brno
(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: Převod křivky z rastru na vektorovou representaci
Vedoucí/školitel VŠKP: Beran Vítězslav, 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

V mém procesu vektorizace se snažím o převod vstupního šedotónového obrazu na vektorový se snahou o co největší podobnost se vstupním obrazem. Vektorizace se provádí pomocí aproximace křivkou, jenže aproximovat lze pouze liniové prvky, tedy křivky v rastru. Musí se proto ze vstupního obrazu tyto liniové prvky vyextrahovat a to dvojím způsobem, podle dvou skupin objektů v obraze. První skupinou jsou objekty tenké, podlouhlého tvaru, ty se nahradí jejich skeletonem. Druhou skupinou jsou objekty rozsáhlé, ty se nahradí jejich konturou. Nalezené linie se pak rozdělí na takové části, které už půjde snadno aproximovat křivkou. Výsledné křivky se už jen vykreslí do výstupu vhodnou rasterizační metodou.

Klíčová slova

Skeletonizace, segmentace, aproximace křivkou, metoda nejmenších čtverců, vektorizace, rasterizace.

Abstract

In my process of tracing I deal with converting an input grayscale image into a vector one trying to keep as big similarity with the input image as possible. Tracing is carried out with the help of curve approximation even if the approximation is possible only with line elements, that is to say the curve in raster. Therefore it is necessary to extract the line elements from the input image. We can do it in two different ways according to two different objects in the image. The first group is represented by thin, ablong objects which are substituted by their skeleton. The second group is represented by large objects which are susbstituted by their contour. The found lines are then divided into such parts which can be easily curve approximated. Resulting curves are then only depicted into the output by suitable raster method.

Keywords

Skeletonization, segmentation, curve approximation, the least squares method, vectorization, rasterization.

Citace

Jiří Král: Převod křivky z rastru na vektorovou representaci, bakalářská práce, Brno, FIT VUT v Brně, 2007.

Převod křivky z rastru na vektorovou representaci

Prohlášení

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

……
Jméno Příjmení
Datum

Poděkování

Poděkování patří vedoucímu mé práce Ing. Vítězslavu Beranovi za obětavost a metodické rady při zpracování bakalářské práce.

Obsah

1 Úvod

S neustálým rozvojem výpočetní techniky a elektronické komunikace vzrůstá i jejich rozsah využití. Jednou z oblastí využití je i elektronické zpravování dokumentů. Dochází k nahrazování papírových dokumentů elektronickou podobou, ať už se jedná o psané nebo o různé obrazové dokumenty. Práce s dokumenty v tištěné podobě bývá častokrát komplikovanější například z důvodu archivace, vyhledávání.

Dnes se již téměř všechny dokumenty vytváří elektronicky, ale co s dokumenty vytvořenými ještě na papíře. Ty se do elektronické podoby nasnímají různými skenovacími zařízeními. Skenováním se sice získá jejich elektronická representace, ale taková representace je pouze rastrová. Na rozdíl od toho dokumenty vytvářené přímo na PC mohou být i ve vektorové podobě. Příkladem jsou počítačové programy pro sazbu textu, kde text je sázen rovnou ve vektorové podobě. Výhodou vektorového formátu od rastrového je v tom, že se s ním dá různě pracovat, jako kupříkladu zvětšování textu, kde písmo po zvětšení zůstává stále vyhlazené. V případě obrazových dokumentů jsou dobrým příkladem použití vektorového formátu mapy, se kterými se dá provádět kromě zvětšování/zmenšování i různé měření vzdáleností.

Nyní si zjednodušeně vysvětlíme princip rastru, vektoru a jejich případné výhody a nevýhody. Rastrová representace je základním způsobem počítačového zobrazení, je používána pro výstup na monitor. Oproti tomu s vektory pracuje jiné výstupní zařízení a to tiskárna. Rastrový obraz je tvořen pixely a pixel je v podstatě jeden bod na monitoru. Rastrový obraz je pak množina bodů, kde každý bod má svou pozici a barvu. Vektorový obraz je popsán matematicky, to znamená, že je složen z matematicky definovaných objektů, jako jsou například přímky, kružnice, elipsy, křivky, atd. Vektorový obraz je tvořen jen matematickým popisem těchto geometrických objektů. Jednoduchou úpravou matematického zápisu, respektive změnou parametrů, se dá geometrický objekt různě upravovat. Z toho vyplývá jasná výhoda vektorové representace. Ovšem, jak již bylo zmíněno, zobrazení na monitoru je rastrové, proto se musí vektorový tvar převést na rastrový, tím se zase o něco zvýší výpočetní náročnost.

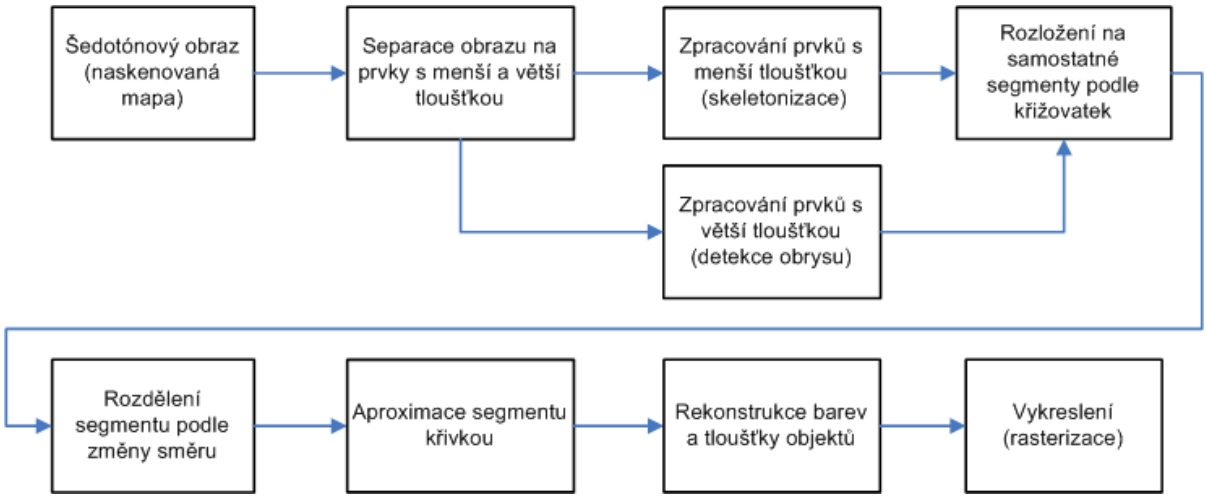
Je mnoho dokumentů jen v tištěné podobě a my je chceme pro snadnější práci přenést do elektronické podoby, do vektorové reprezentace, která sčítá řadu již zmíněných výhod. Jenže jak tyto dokumenty dostat do vektorové podoby, když naskenováním získáme pouze rastrovou reprezentaci. Touto otázkou jsme se dostali k cíli mé práce a to je takzvaná vektorizace.

Pro převod do vektorové podoby budu používat křivek a to proto, že dovedou reprezentovat libovolné tvary, ne jen rovné linie jako přímka, ale i například ručně nakreslené čáry.

Smyslem mé práce je vstupní rastrový obraz převést na vektorový pomocí křivek tak, aby se výsledný obraz co nejvíce podobal originálnímu obrazu. Kromě tvarů se pokusím i o rekonstrukci barev, respektive jasu, jelikož pro zjednodušení pracuji z šedotónovým obrazem. Vektorizovaný

obraz je kvůli výslednému zobrazení zpětně rasterizován. Vektorový popis je možné dále využít, k různým úpravám měřítka, k rekonstrukci apod., to už ale není v mé práci obsaženo. Moje řešení je testováno především na různých naskenovaných mapách, protože se domnívám, že mapy jsou nejlepším typem obrazů pro můj úkol. Obsahují totiž různé typy objektů, jako tenké podlouhlé prvky (cesty, potoky), různé značky a rozsáhlé objekty (oblasti zalesnění). A každý z těchto objektů vyžaduje různý způsob zpracování.

Nyní seznámím čtenáře s jednotlivými etapami a průběhem funkce programu (viz obrázek 1). Po načtení šedotónového obrazu následuje rozdělení obrazu na dva typy objektů, jenž se liší způsobem zpracování. V mapě jsou tenké podlouhlé prvky (cesty, potoky), které je možno zpracovat nalezením jejich středové linie (skeletonizace) a rozlehlé prvky s větší tloušťkou (značky, oblasti zalesnění), kde by nalezení středové linie způsobilo značnou deformaci, proto se hledá obrys takovýchto objektů. Další zpracování už je společné pro obě skupiny, výsledek, tedy linie (ať už středová nebo obrysová) se rozloží na samostatné části tzv. segmenty podle křižovatek a koncových bodů. Obraz je tím rozložený na jednoduché čáry o tloušťce jedna, taková čára by se už dala aproximovat křivkou, ale mohlo by dojít ke značnému zkreslení čáry, jelikož čára může být různě zakřivená a může mít i různá zalomení a to se jednou křivkou nedá dost přesně nahradit. Tudíž se takovýto segment rozdělí na více segmentů a každý z nich bude nahrazen jednou křivkou. Rozdělení segmentu na více segmentů podle zakřivením se provádí na základě směru gradientu. Poté už je segment připraven na aproximaci křivkou, k čemuž se využívá metody nejmenších čtverců, tu si ale popíšeme až v kapitole 5.2. Každý segment je popsán funkcí křivky. Z původního obrazu se pro každou vypočítanou křivku určí hodnota jasu a tloušťka. Poté se už jen křivka vykreslí pomocí rastrovacího algoritmu a tím je celý postup završen.



Obrázek 1: Blokové schéma celého algoritmu.

2 Separace obrazu na prvky s menší a větší tloušťkou

Samotná metoda nahrazení křivkou pracuje s řetězcem pixelů o tloušťce jedna. Potřebujeme proto vyextrahovat ze vstupního obrazu řetězce pixelů o jednotkové tloušťce. Způsobů jak z obrazu získat jen jednotkové řetězce pixelů je několik. Jako vhodná varianta se jeví metoda nalezení středové linie. Jedná se o metodu nalezení skeletonu (kostry) obrazu. Ovšem použití této metody je vhodné zejména pro prvky s menší a rovnoměrnou tloušťkou po celé délce prvku, jedná se o prvky typu tenká dlouhá čára. To jsou takzvané tenké prvky, v případě mapy se může jednat o cesty, silnice, řeky. Ale použití skeletonizace na prvky, jejichž tloušťka je významně větší a proměnná, už tak vhodné být nemusí. Jedná se o u různé značky, oblasti zalesnění, nazvěme je tlusté prvky. Nahrazením takového prvku středovou linií dojde ke značné ztrátě informací a k deformaci prvku. Místo hledání středové linie zvolíme metodu pro nalezení kontury (obrysu) prvku.

Důvod proč obraz separovat na tyto dvě skupiny je již znám, ale jakým způsobem tyto prvky rozdělit. Potřebujeme je separovat tak, aby nedošlo k jejich porušení, ale aby byl separován celý prvek. Využijeme jejich barvy, respektive jasu u šedotónového obrazu. V originálním obraze má každý prvek svou hodnotu jasu a toho využijeme. Detekcí na základě jasové hodnoty zajistíme i zachování celistvosti detekovaného prvku.

Prvně obraz vyhladíme jednoduchým lineárním filtrem, abychom potlačili šum v obraze vzniklý skenováním. Dále rozdělíme obraz na jasové skupiny, tím od sebe částečně oddělíme jednotlivé prvky. V každé skupině budeme hledat objekty na základě tloušťky a objekty s větší tloušťkou oddělíme (separujeme) od původního obrazu.

2.1 Detekce tloušťky objektů

Máme obraz jen s objekty dané jasové skupiny a z něj chceme vyextrahovat objekty s danou tloušťkou. K tomu se využije výpočet vzdálenostní mapy, který každému pixelu v obraze říká jakou má nejkratší vzdálenost ke krajnímu (nulovému) pixelu (viz další kapitola). Ze vzdálenostní mapy se získá skeleton (viz kapitola 3), který je v podstatě tvořen vrcholy (největšími hodnotami) vzdálenostní mapy. Tyto vrcholy už udávají nejmenší možnou vzdálenost k okolí objektu, tedy tloušťku objektu v daném místě. Pro zjištění tloušťky budeme procházet skeleton každého objektu a číst jeho hodnoty vzdálenostní mapy. Na závěr stačí tyto hodnoty pro každý objekt zprůměrovat a to je již směrodatná hodnota, na základě které se rozhodneme, zdali objekt patří mezi tenké nebo tlusté prvky.

2.1.1 Výpočet vzdálenostní mapy

Vzdálenostní mapa pracuje nad binárním obrazem, kde bod okolí má hodnotu 0 a bod objektu má hodnotu 1. Jedná se o dvouprůchodovou metodu. V každém bodě počítá nejkratší vzdálenost k pozadí. Následný způsob výpočtu vychází z metody P. Altmana [2]. Zvolme vzdálenostní metriku mezi dvěma body:

- povolíme pouze horizontální/vertikální krok s váhou $w_1 = 1$, dostaneme „městskou metriku“ d_1,
- pokud navíc povolíme i diagonální krok s váhou $w_1 = 1$, dostaneme „šachovnicovou metriku $d_{1,1}$,
- lepší aproximaci vzdálenosti získáme změnou vah na $w_1 = 3$ (pro horizontální/vertikální krok) a $w_2 = 4$ (pro diagonální krok), tuto metriku označme jako $d_{3,4}$,
- ještě lepší aproximaci získáme, pokud povolíme navíc ještě „skok šachovým koněm a použijeme váhy $w_1 = 5$ (pro horizontální/vertikální krok), $w_2 = 7$ (pro diagonální krok) a $w_3 = 11$ (pro skok koněm), tuto metriku označme jako $d_{5,7,11}$.

Pro svou implementaci jsem zvolil metriku d_1.

Slovní popis: Každému bodu objektu přiradí nejmenší hodnotu okolí podle zvolené metriky a podle zvoleného kroku.

Označme aktuální bod objektu p a i-té okolí bodu p jako $n_i(p)$:

n_2	n_3	n_4
n_1	p	n_5
n_8	n_7	n_6

Obrázek 2: Okolí n_i bodu p.

1. **Dopředný průchod:** Výpočet probíhá po řádcích od levého horního do pravého dolního rohu obrazu.
$$f_1(p) = \min[n_1(p) + w_1, n_3(p) + w_1]$$

2. **Zpětný průchod:** Výpočet probíhá po řádcích od pravého dolního do levého horního rohu obrazu.
$$f_2(p) = \min[n_5(p) + w_1, n_7(p) + w_1]$$

Příklad výpočtu vzdálenostní mapy s metrikou d_1:

0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	1	2	2	2	2	2	2	1	0
0	1	2	3	3	3	3	3	2	0
0	1	2	3	4	4	4	4	3	0
0	1	2	3	4	5	5	5	4	0
0	0	1	2	3	4	5	6	0	0
0	0	0	0	0	0	0	0	0	0

0	0	0	0	0	0	0	0	0	0
0	0	1	1	1	1	1	1	0	0
0	1	2	2	2	2	2	2	1	0
0	1	2	3	3	3	3	2	1	0
0	1	2	3	3	3	3	2	1	0
0	1	2	2	2	2	2	2	1	0
0	0	1	1	1	1	1	1	0	0
0	0	0	0	0	0	0	0	0	0

Obrázek 3: Výpočet vzdálenostní mapy. Vlevo je výsledek po prvním průchodu, vpravo po druhém průchodu.

3 Zpracování prvků s menší tloušťkou

V této kapitole se budu zabývat přípravou tenkých prvků na následnou vektorizaci. Abych mohl provést vektorizaci, musím z obrazu získat množinu linií, což jsou řetězce pixelů o tloušťce jedna. Tyto linie se pak rozdělí na takové části, které už lze nahradit křivkou. Způsobů nalezení linií je několik, jelikož se bavíme o tenkých prvcích, bude vhodné použít metody nalezení středových linií, chcete-li skeletonů objektů.

3.1 Metody nalezení skeletonu

Skeleton je kostra, nebo-li středová osa objektu, zachovávající topologii vstupního obrazu. Využívá se pro vyjádření podlouhlých objektů v obraze. Detailnější popis je možné najít v [1]. Představíme si několik různých přístupů hledání skeletonu:

Sekvenční ztenčování: Tato metoda lze přirovnat k „loupání cibule“. Proces se provádí opakovaně, při čemž v každém kroku se odloupne vrstva o tloušťce jedna a to pomocí posloupnosti strukturních elementů, které zajišťují zachování souvislosti objektů. Odstraňování vrstev se děje tak dlouho, dokud není dosaženo linie, poté proces skončí. Obvykle se jedná o homotopické ztenčování, s využitím strukturních elementů z Golayovy abecedy.

Metoda maximálních disků: Metoda je založena na výpočtu vzdálenostní mapy. K nalezení skeletonu dojde v konstantním počtu průchodů. Také zachovává souvislosti objektů, jedná se o nejrychlejší a o nejčastěji používanou metodu. Použil jsem ji ve svém programu a bude podrobněji probrána v následující kapitole.

Vpisování kruhů: Využívá středů maximálních vepsaných kruhů. Příliš výpočetně náročné, nezachovává souvislost, skelet bývá tloušťky větší jak jedna. V praxi se téměř nepoužívá.

V koutkové reprezentaci: Prvně se oblasti bezeztrátově komprimují do koutků, poté se vypočítá skelet a to vpisováním maximálních obdélníků přímo v komprimovaných datech.

3.2 Metoda nejmenších disků

Metoda pracuje s vzdálenostní mapou (kapitola 2.1.1). Vychází z algoritmu pro nalezení středové linie v práci [2] s výjimkou způsobu odstranění krátkých větví, který je vyloženě mým vlastním postupem. Využijme již zavedeného označení pro bod objektu p a i-té okolí bodu p jako $n_i(p)$,

kde $i \in \{1,...,8\}$. Váhu mezi dvěma pixely označme w_k. Označme F jako množinu pixelů objektu a B jako množinu pixelů okolí. V následujícím textu budeme bod p chápat jednak pro pozici pixelu a jednak pro jemu přiřazenou vzdálenost. Libovolný bod p ze vzdálenostní mapy můžeme chápat jako střed disku padnoucího do F a vzdálenost p jako poloměr disku. Pixel nazvěme jako střed maximálního disku, pokud jemu příslušející disk je maximální, tedy neexistuje jiný disk, který ho celý obsahuje. Středy maximálních disků budeme označovat jako *cmd* (center of maximal disk).

3.2.1 Nalezení středové linie

Středová linie je tvořena středy maximálních disků a propojovacími pixely. V prvním průchodu vzdálenostní mapy se určí *cmd* a pak se pro každý *cmd* naleznou propojovací pixely.

Nalezení středů maximálních disků: Z definice víme, že bod p je *cmd*, pokud neexistuje jiný disk, který ho celý obsahuje. Při zjišťování, zda-li bod p je *cmd* nemusíme kontrolovat všechny disky v okolí, to by bylo výpočetně příliš náročné, ale stačí porovnat hodnoty bodů v okolí bodu p. Pixel p označíme jako *cmd* pokud platí, že $n_i(p) < p + w_k$ pro $i \in \{1,...,8\}$. Hodnota w_k je závislá na vzájemné poloze $n_i(p)$ a p ($w_k = 3$, pokud i je liché nebo $w_k = 4$, pokud i je sudé).

Hledání propojovacích pixelů: Abychom zachovali topologii původního obrazu, musí být skeleton spojitý, bude tedy nutné propojit jednotlivé komponenty. Prochází se každý bod *cmd* nalezený v prvním průchodu a hledá se jeho soused podle následujícího principu. V každém bodě *cmd* vybereme souseda s největším kladným gradientem a ten bude propojovacím pixelem. Gradient souseda $n_i(p)$ se vypočítá jako $grad_i = [n_i(p) - p]/w_k$. Může nastat situace, že bude vypočítána stejná hodnota gradientů pro více sousedů, v tom případě se zvolí za propojovací pixel ten soused, který je v horizontálním/vertikálním směru vůči *cmd*. Tedy takový bod $n_i(p)$, kde i je liché.

3.2.2 Úprava středové linie

Již můžeme tvrdit, že jsme nalezli skeleton obrazu, ovšem skeleton zatím není příliš použitelný pro další zpracování, poněvadž ne ve všech místech má jednotkovou tloušťku, dále kvůli šumu je skeleton klikatý a na koncích roztřepený. Toto jsou špatné vlastnosti skeletonizace a my se je pokusíme v následujícím textu alespoň částečně eliminovat.

Redukce středové linie na jednotkovou šířku: To, že skeleton není jednotkové tloušťky je způsobeno výpočtem vzdálenostní mapy, která umožňuje existenci více cmd vedle sebe. V tomto kroku budeme taková místa ztenčovat na tloušťku jednoho pixelu. To musíme realizovat operací zachovávající topologii. Tato operace vychází z toho, že nesmí docházet ke zkracování skeletonu na koncích. Celá operace se provádí v několika krocích. V prvním kroku nalezneme body skeletonu, které jsou interní, tedy ve všech horizontálních/vertikálních směrech sousedí s body skeletonu. V dalším kroku procházíme body skeletonu, které nebyly označeny jako interní a zrušíme jejich přiřazení ke skeletonu, pokud je splněna následující podmínka:

$n_i(p)$ a $n_{i+2}(p)$ jsou body skeletonu a $n_{i+5}(p)$ není bodem skeletonu pro $i \in \{1,3,5,7\}$. A zároveň existuje alespoň jeden horizontální nebo vertikální soused bodu p, který není bodem skeletonu.

Odstranění krátkých koncových větví: Tímto procesem se budeme snažit odstranit roztřepení na koncích skeletonů, které je neblahým rysem skeletonizace, a „krátké fousky“, které jsou zapříčiněny šumem na okrajích prvků a rovněž redukcí na jednotkovou šířku.

V prvním kroku nalezneme body skeletonu p, které jsou jeho koncovými body. Takové body budeme hledat podle podmínky:

Bod p je koncový, pokud má právě jednoho souseda $n_i(p)$ nebo, pokud má právě dva sousedy $n_i(p)$ a $n_{i+1}(p)$ pro $i \in \{1,...,8\}$.

V dalším kroku už budeme hledat větve skeletonu, které chceme odstranit. Začneme vždy procházením skeletonu od nalezeného koncového bodu z předchozího kroku. Budeme procházet po bodech skeletonu, sousedních bodech aktuálního bodu. Při průchodu si vždy aktuální bod uložíme do pomocného řetězce a vyznačíme ho, abychom jej už dále nedetekovali. Průchod skeletonem se bude provádět tak dlouho, dokud nenarazíme na konec skeletonu nebo na křižovatku. Konec skeletonu je takový bod, který už nemá žádné sousedy $n_i(p)$. Křižovatka je takový bod, který má dva sousedy, jenž spolu navzájem nesousedí ani ve vertikálním ani v horizontálním směru, což lze definovat takto:

Musí být splněno, že sousedními body bodu p nesmí být $n_i(p)$ a zároveň ani $n_{i+1}(p)$ pro $i \in \{1,...,8\}$.

Nebo je křižovatka takový bod, který má tři a více sousedů.

Pokud je průchod ukončen nalezením koncového bodu, tak se jedná o samostatný skeleton, který chceme v obraze zachovat, vyprázdníme řetězec pixelů skeletonu a skončíme. Pokud je průchod ukončen nalezením křižovatky, tak se jedná o koncovou větev skeletonu a ta by mohla být nežádanou větví. Zjistíme délku takovéto větve, tedy počet pixelů uložených v řetězci a pokud je délka menší nebo rovna námi zadané délce, jedná se o nežádoucí větev a tu z obrazu pomocí řetězce pixelů odstraníme a skončíme.

Vyhlazení skeletonu: Ze skeletonu už jsou eliminovány „fousky“ a roztřepené konce, ale ještě zbývá vyhladit klikaté části skeletonu. To se provede přesunutím některých bodů skeletonu na vhodného souseda.

Nechť bod p je bodem skeletonu a má právě dva sousedy, které jsou též součástí skeletonu, $n_i(p)$ a $n_{i+2}(p)$ pro $i \in \{2,4,6,8\}$ a nechť $n_{i+1}(p) \in F$. Potom zrušíme původní bod skeletonu p a novým bodem skeletonu zvolíme $n_{i+1}(p)$.

3.3 Rozdělení skeletonu podle významných bodů

Původní obraz je nyní nahrazen jeho skeletonem. Vektorizační algoritmus pracuje s řetězcem pixelů, který tvoří samostatnou linii. Musíme proto skeleton rozložit na samostatné řetězce pixelů. Rozložení se provede pomocí významných bodů, což jsou křižovatky a koncové body skeletonu. Každý řetězec bude začínat a končit významným bodem. V prvním kroku se naleznou tyto významné body a v dalším kroku se budou části skeletonu, dané významnými body, ukládat do samostatných řetězců. Těmto řetězcům pixelů budeme říkat segmenty.

3.3.1 Detekce významných bodů

Algoritmus postupně prochází všechny body skeletonu a testuje je, zda-li jsou koncovými body nebo křižovatkami, pokud ano, tak takovýto bod v obraze označí.

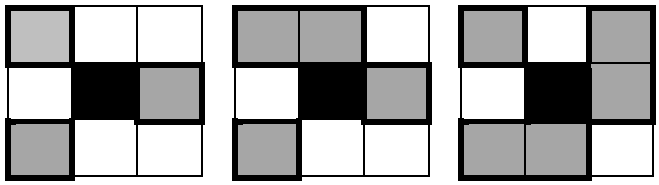
Ověřování, zda-li je bod p koncový se bude provádět stejně jako v případě odstraňování krátkých větví v kapitole 3.2.2. Tedy bod skeletonu p je koncový, pokud má pouze jednoho souseda. Pokud má dva sousedy tak je koncový, pokud sousedními body jsou $n_i(p)$ a $n_{i+1}(p)$ pro $i \in \{1,...,8\}$.

Ověřování, zda-li je bod p bodem křižovatky je již o něco složitější, vychází z počtu sousedních bodů:

- Méně jak tři: Nemůže se jednat o křižovatku v žádném případě.
- Více jak dva a méně jak šest: Může a nemusí se jednat o křižovatku, musí se testovat podle jiných kritérií.
- Více jak pět: O křižovatku se jedná v každém případě.

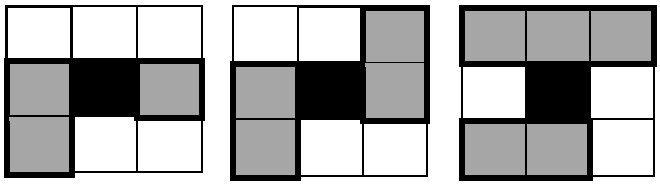
Jestliže má bod p tři až pět sousedů, bude nás zajímat jejich rozložení. Aby se jednalo o křižovatku, musí mít bod p alespoň tři skupiny sousedních pixelů. Skupina je tvořena pixely, které spolu sousedí v horizontálních/vertikálních směrech. Skupiny mezi sebou v horizontální/vertikálních

směrech nesousedí. Různé typy křižovatek jsou znázorněny na obrázku 4. Na obrázku 5 jsou znázorněny situace, kdy se o křižovatky nejedná. Tučným obrysem jsou znázorněny jednotlivé skupiny.



(a) (b) (c)

Obrázek 4: Křižovatky: (a) 3-okolí, 3 skupiny. (b) 4-okolí, 3 skupiny. (c) 5-okolí, 3 skupiny.



(a) (b) (c)

Obrázek 5: Sousední body bodu p : (a) 3-okolí, 2 skupiny. (b) 4-okolí, 2skupiny. (c) 5-okolí, 2 skupiny.

3.3.2 Detekce samostatných částí skeletonu

Nyní máme v obraze skeletonu vyznačeny významné body. Budeme procházet skeleton od každého významného bodu a to po sousedních pixelech. Každý projitý bod prohlásíme za detekovaný, abychom zabránili opakovanému průchodu již projitých bodů. Aktuální bod rovněž uložíme do segmentu (pro připomenutí se jedná o samostatný řetězec pixelů). Průchod skeletonu a ukládání do segmentu bude trvat tak dlouho, dokud nenarazíme na další významný bod. Po jeho nalezení aktuální segment uložíme jako kompletní a pokračujeme v průchodu od dalšího významného bodu s ukládáním do nového segmentu.

Po skončení průchodu skeletonu již máme množinu segmentů, kde každý segment začíná a končí významným bodem. Jenže nemáme v segmentech uloženy uzavřené objekty, tedy kružnice a to proto, že neobsahují žádné významné body, ani křižovatky, ani koncové body. Takovéto objekty se dají detekovat dalším průchodem, protože obraz se skeletonem má již procházené body vyznačené. Tudíž pro nalezení kružnic budeme procházet nevyznačené pixely. Protože kružnice nemají významné body, prohlásíme za významný bod první nalezený bod kružnice. Následný průchod bude stejný jako v případě neuzavřených objektů, akorát že významný bod je jen jeden a proces skončí po jeho nalezení. Tímto je celý proces hotov, vstupní obraz už obsahuje jen vyznačené pixely.

4 Zpracování prvků s větší tloušťkou

Podobně jako v předchozí kapitole i zde potřebujeme nalézt linie objektů, tedy řetězce pixelů pro umožnění vektorizace. Zatímco u tenkých prvků se použití metody pro nalezení středové linie přímo nabízelo, protože skeletonizace je vhodná pro podlouhlé tenké prvky, zde u rozměrnějších objektů by skeletonizace potlačila mnoho informací o objektu. Nalezení středové osy v tomto případě nemá význam. Proto se musíme zaměřit na jiné metody, po jejichž použití by byly zachovány důležité informace o objektu. Jedná se metody hledání hran a obrysů v obraze. V následujícím textu se s nimi ve stručnosti seznámíme.

4.1 Hranové detektory

Slouží k redukci dat, získání důležitých rysů obrazu. Hrana je tvořena pixely obrazu, kde se mění prudce hodnota jasu. V každém pixelu je definován gradient (podrobně v kapitole 5.1.1), který udává největší změnu hodnoty jasu, tedy velikost (modul) gradientu a směr této největší změny. Hranami jsou právě pixely s největšími moduly gradientu. Následovně si představíme nejpoužívanější detektory hran, jedná se jen o přehled, jednotlivým metodám se podrobně věnuje [1].

1. Jednoduché konvoluční masky aproximující derivace obrazové funkce

Jedná se o operátory aproximující první derivaci, jedná se o detekci extrémů. Tyto filtry analyzují jen malé lokální okolí, provádí konvoluci s jádrem daného hranového operátoru. Umožňují detekovat i směr hrany a to použitím více jader pro každý směr. Jsou velmi závislé na velikosti objektu v obraze a také mají značnou citlivost na šum.

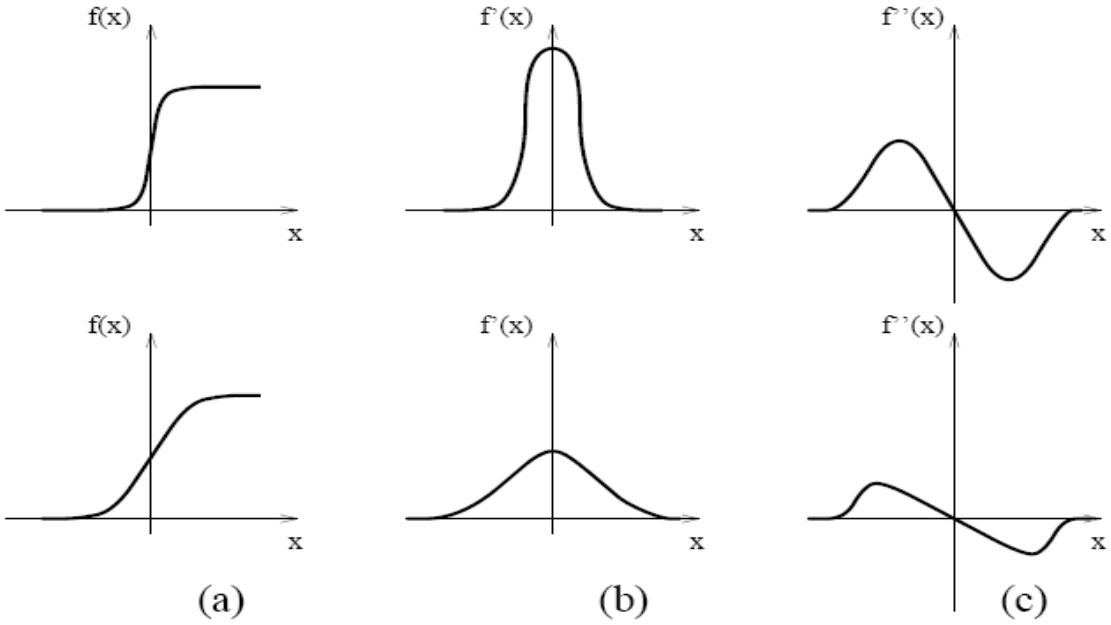
Robertsův operátor: Jedná se o nejstarší a velmi jednoduchý operátor, používá málo pixelů k výpočtu gradientu, proto je velká citlivost na šum. Neumí detekovat směr gradientu.

Sobelův operátor: Používá pouze dvě masky, tudíž detekuje pouze dva směry, horizontální/ vertikální hrany.

Operátor Kirschův, Robinsonův a Prewittové: Podobně jako Sobelův operátor gradient je vypočítáván v okolí 3x3 ale tentokrát pro osm směrů. Vybrána je jedna maska z osmi a to ta, které odpovídá největší modul gradientu.

2. Hrany jako průchod nulou druhé derivace obrazové funkce

První derivace obrazové funkce nabývá maxima v místě hrany na rozdíl od toho druhá derivace v místě hrany protíná nulovou hodnotu. Tato situace je znázorněna na obrázku 6, kde (a) ukazuje skokovou hranu, (b) její první derivaci a (c) její druhou derivaci. Hledat směr hrany v místě průchodu nulou je mnohem spolehlivější, než na maximu v případě první derivace.



Obrázek 6: Skoková hrana a její derivace.

Laplacián: Je invariantní vůči otočení a udává jen velikost hrany a ne její směr. Nevýhodou je, že je příliš citlivý na šum a že způsobuje dvojitou odezvu na hrany odpovídající tenkým liniím v obraze.

LoG – Marr a Hildreth hranový operátor: Jedná se o Laplacián obrazu vyhlazeného Gaussem. Jde o velice stabilní a robustní metodu měření změn v obraze a jejich detekcí. Lokalizuje nulu v místě hrany a při výpočtu se používá většího okolí pixelu.

Cannyho detektor: Základní myšlenka je taková, že skokové hrany lze hledat filtrem. Detektor je optimální pro skokové hrany vzhledem ke třem kritériím:

1. Detekční kritérium: Požaduje, aby významné hrany nebyly přehlédnuty a aby nebyly vícenásobné odezvy na jednu hranu.
2. Lokalizační kritérium: Požaduje, aby rozdíl mezi skutečnou a nalezenou polohou hranou byl minimální.

3. Jedinečná odezva: Zajišťuje, aby detektor nereagoval na jednu hranu v obraze vícenásobně. Toto je již částečně zajištěno prvním kritériem, ale tento požadavek je zaměřen zejména na zašuměné a nehladké hrany.

Cannyho detektor je realizován různými způsoby, které se snaží co nejlépe vyhovět následujícím podmínkám:

- využití operátoru první derivace Gaussovské funkce, jedná se konvoluci s 2D Gaussiánem a o derivaci ve směru gradientu,
- potlačení nemaximálních hodnot (non-maxima suppresion), ve směru gradientu potlačí ty pixely, které nejsou lokálním maximem
- prahování s hysterezí, hystereze umožní nadetekovat i ty části hran, které jsou pod prahovou hodnotou.

Houghova transformace: Používá se pro reprezentaci čar a rovinných křivek. Provádí transformaci z Kartézského souřadnicového systému do polárního. Metoda je velice robustní a není citlivá na porušená data ani šum. Nevýhodou ovšem je výpočetní náročnost.

4.2 Detekce obrysů

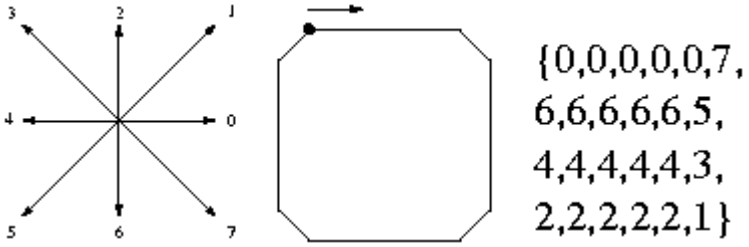
Pro extrakci linií z tlustých prvků by se dalo využít hranových detektorů, které jsme si připomněli v předešlé kapitole. Jenže aplikace jakéhokoliv hranového detektoru je v tomto případě zbytečná, protože námi nadetekované tlusté prvky jsou rozděleny podle barev respektive jasu a uloženy každý samostatně. Každý takovýto prvek má tedy svou hodnotu jasu. Aplikace hranového filtru na tyto prvky by nám vrátila stejně jen obrysové body.

Obrysové body, jsou takové body objektu, které sousedí s okolím objektu. A k nalezení obrysových bodů není třeba hranových filtrů, ale mnoho jednodušších způsobů. Jedním z nich je nalezení obrysu prvku pomocí morfologické operace eroze [1]. Eroze se provádí nad binárním obrazem, tedy jde o binární erozi. Eroze se dá zjednodušeně pospat jako metoda „loupaní cibule“, kdy se v jednom kroku odstraní horní vrstva objektu. Pro nalezení obrysu se tedy na binární obraz aplikuje eroze, poté se erodovaný obraz odečte od původního binárního obrazu, tím získáme obrys, tedy vrstvu erodovanou v jednom kroku.

Každý námi samostatně detekovaný prvek je vždy tvořen jedním vnějším obrysem a může být tvořen několika vnitřními obrysy. Vnější obrys je vnější okolí prvku a vnitřní obrys je obrys děr prvku. Prvek může a nemusí tyto vnitřní díry obsahovat. Chceme, aby každý samostatný obrys prvku byl reprezentován právě jedním segmentem. To proto, abychom zachovali souvislost prvků obrysu. K nalezení obrysových prvků použijeme funkci obsaženou v knihovně OpenCV [5] a to funkci FindContours, která je pro nás vhodná právě proto, že kromě toho že každý obrys prvku uloží zvlášť,

zachová souvislost prvků jednotlivých obrysů. Každý samostatný obrys prvku je uzavřenou oblastí, tedy kružnicí a požadujeme, aby body obrysu byly uloženy jako body kružnice, v takovém pořadí, že první bod obrysu je i zároveň poslední bodem obrysu. Funkce FindContours toto všechno splňuje. Pokud bychom vytvořili obrysové body ručně pomocí operace eroze a odečtením a poté použili metodu z kapitoly 3.3 pro nalezení segmentu podle významných bodů, tak by nastal následný problém. V obrysech by detektor významných bodů našel nežádoucí křižovatky a podle nich by pak obrysy rozděloval do segmentů. Nastalo by to, že z důvodu křižovatek by obrys, tedy uzavřená oblast byla prezentována více segmenty a to ještě v různém pořadí. Z takto vytvořených segmentů v pozměněném pořadí už by nebylo možné vytvořit uzavřený polygon.

Zdůvodnil jsem použití funkce FindContours, nyní si popíšeme její princip. Jak již bylo řečeno, funkce ukládá každý obrys samostatně a zachovává souvislost obrysových bodů. Nalezne první bod obrysu a pokračuje v procházení obrysových bodů po sousedech. Aktuální bod vždy uloží do vlastní struktury. Ukládání provádí ve tvaru takzvaného Freemanova chain kódu. Jedná se o způsob reprezentace bodů, kde první bod je uložen jako souřadnice bodu a následující body jsou udávány číslem od 0 do 7. Toto číslo vyjadřuje směr, ve kterém následující bod sousedí s aktuálním bodem (viz obrázek 7).



(a) (b) (c)

Obrázek 7: Chain kód: (a) směry všech osmi sousedních bodů, (b) příklad objektu, (c) chain kód representující objekt.

Průchod skončí, jakmile dojde k počátečnímu bodu, poté začne hledat stejným způsobem další obrysy a ty ukládá vždy do nových řetězců. S obrysem popsaným pomocí chain kódu by se nám příliš dobře nepracovalo, naštěstí funkce FindContours umí sama tento zápis převést do normální representace, tedy do tvaru, kde je každý bod popsán svými souřadnicemi. Výsledné řetězce jednotlivých obrysů uložíme do námi zavedených segmentů. Nyní máme každý samostatný obrys reprezentovaný jedním segmentem. S tím už můžeme pracovat.

4.3 Vytvoření polygonů

Máme obrys, který representuje původní tlustý prvek. Jenže v původním obraze se jednalo o vyplněnou oblast, ne jen o obrys, tudíž potřebujeme obrys vyplnit barvou/jasem odpovídající

originálu. Obrys se bude vektorizovat a vektorizovaný obrys pak vyplníme. K tomu ale potřebujeme mít uzavřenou oblast, proto budeme vytvářet s každého segmentu reprezentující samostatný obrys uzavřený polygon. Prvně se musí každý vstupní segment vektorizovat, vektorizací se budu důkladně zabývat v kapitole 5. Zde nastíním jenom postup, abychom mohli na výsledek vektorizace navázat vytvořením polygonů.

Každý vstupní segment se rozdělí na menší segmenty podle zakřivení vstupního segmentu. Výsledné segmenty jsou uloženy do množiny segmentů v takovém pořadí, aby zůstala zachovaná souvislost mezi jednotlivými body obrysu. Každý takto rozdělený segment se pak může aproximovat křivkou, poté zpětně rasterizovat, tedy vypočítat body ležící na křivce. Ve výsledku máme pro každý vstupní segment, reprezentující samostatný obrys, množinu segmentů s vypočítanými body křivky. Máme množinu segmentů, které na sebe navazují a zachovávají pořadí bodů obrysu. Body všech segmentů uložíme do pole ve stanoveném pořadí. Pole už udává všechny vypočtené body křivek pro cely vstupní segment. Tento celý proces budeme aplikovat pro každý vstupní segment daného prvku a budeme ukládat vždy do nového pole. Výsledkem celé operace je, že máme několik polí, kde každé obsahuje vypočtené body vektorizováného obrysu. Použijeme další funkci z knihovny OpenCV a to funkci FillPoly, která s jednotlivých polí vytvoří uzavřený polygon a to propojením jednotlivých vrcholů (bodů pole) úsečkami. Vytvořené uzavřené polygony sama i vyplní a to pomocí řádkového vyplňovacího algoritmu, ten bude probrán v následující kapitole.

4.4 Vyplňování oblastí

Zde uvedu základní metody vyplňování viz literatura [7] a zaměřím se především na metodu řádkového vyplňování, použitou v mém algoritmu.

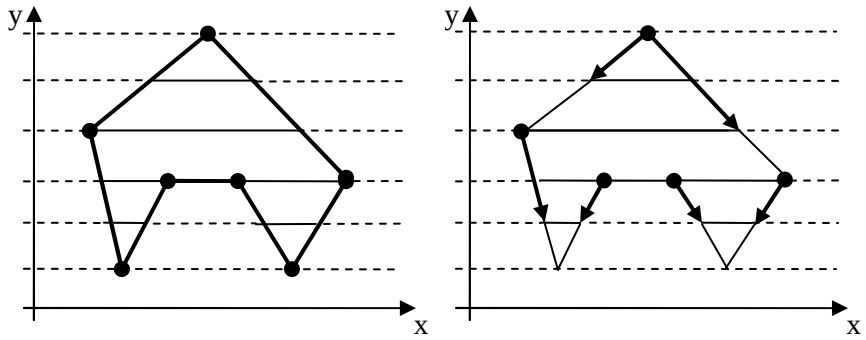
Vyplňování oblastí je proces, při kterém se hledají všechny vnitřní body oblastí a nastavují se na určitou hodnotu. Vyplňovaná oblast musí být samozřejmě uzavřená. Rozlišují se dva základní způsoby popisu hranice oblastí:

1. Geometrický popis

Hranice jsou popsány množinou úseček, popřípadě křivek.

Řádkové vyplňování: Jedná se o základní metodu. Každým řádkem rastru je vedena přímka a jsou vypočítávány její průsečíky s hranicí oblasti. Nalezené průsečíky se seřadí podle souřadnice x. Dvojice průsečíků mezi lichým a sudým definují úsek uvnitř oblasti pro obarvení pixelů. Hledání vnitřních úseček probíhá shora dolů. V případě, že procházející řádek má nulovou délku (viz 1. řádek na obrázku 8), dojde k vykreslení úsečky délky nula, což by měl být jeden pixel. Pokud řádek protíná vodorovnou hranou (viz 4. Řádek na obrázku 8), má s ní nekonečně mnoho průsečíku, proto se

takovéto hrany vylučují. Další problém je v řádku, který protíná lichý počet průsečíku, to bývá způsobeno lokálním maximem v ose x (viz 3. řádek na obrázku 8). Standardním řešením bývá zkrácení takové hranice o jeden pixel zdola ve směru osy y. Zkrácení hranice nemá na algoritmus funkční vliv, ba naopak přispívá ke zrychlení algoritmu, z důvodu zmenšení počtu průsečíků, proto se zkracování hranic aplikuje systematicky na všechny hrany.



Obrázek 8: Příklad řádkového vyplňování.

Inverzní řádkové vyplňování: Každým řádkem rastru je vedena přímka zleva od průsečíku až po maximální souřadnice osy x. Každá křivka hranice se zpracovává zvlášť. Při vykreslování bodů rastru přímky se hodnoty invertují. A stejně jako u klasického řádkového vyplňování se vypouštějí vodorovné hrany a zkracují se hranice zdola ve směru osy y o jeden pixel. Na začátku se provede vynulování šablony v bufferu. Tento algoritmus má lineární časovou složitost a to proto, že není třeba třídit průsečíky, zato paměťová náročnost je z důvodu použití bufferu vyšší.

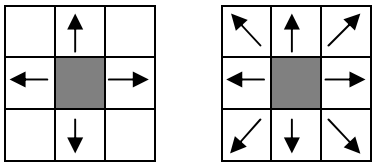
Pinedův algoritmus: Slouží pro vyplňování konvexních oblastí. Oblast vyplňování je dána průnikem polorovin jednotlivých hran hranice. Výhodou tohoto algoritmu je snadná hardwarová implementace.

2. Rastrový popis

Hranice jsou popsány množinou bodů.

Semínkové vyplňování: Mějme danou oblast, která je ohraničena uzavřeným polygonem. Dále potřebujeme jeden startovací bod, od kterého má vyplňování začít, tento bod si jednoduše zvolí sám uživatel pomocí myši. Podle okolí startovního bodu algoritmus rozpozná hodnoty pixelů oblasti, které má vyplňovat, a pokud narazí na pixely s jinou hodnotou, tak se jedná o hranice oblasti a vyplňování v tom směru skončí. Podle okolí bodu, které algoritmus prohledává, máme dva způsoby vyplňování, jsou zobrazeny na obrázku 9, kde (a) je metoda používající čtyř-směrové vyplňování, a (b) metoda

používající osmi-směrové vyplňování. Způsob vyplňování se volí podle typů ohraničení oblasti, buďto čtyř-směrové ohraničení nebo osmi-směrové ohraničení.



(a) (b)

Obrázek 9: Způsoby vyplňování.

5 Vektorizace segmentu

Konečně se dostávám k samotné vektorizaci, stěžejní kapitole celé práce, jak ostatně vyplývá z názvu. Po celou dobu jsem se zatím veskrze zabýval pouhou přípravou vektorizace. Řečeno ve zkrácené formě, v předchozích kapitolách jsem se snažil z původního obrazu vyextrahovat segmenty, které už lze vektorizovat. Segment není nic jiného, než řetězec pixelů representující jednotkovou linii, tu si můžeme představit jako libovolně dlouhou čáru o jednotkové tloušťce.

Vektorizace je převod rastrové reprezentace na vektorovou representaci. Rastrová representace obrazu je množina bodů obrazu, zatímco vektorová representace obrazu je množina geometrických prvků, které jsou tedy matematicky popsány. Vektorizace je nahrazení množiny pixelů, reprezentující určitý objekt v obraze, co nejlépe odpovídajícím geometrickým prvkem. Geometrickými prvky bývají nejčastěji přímky, elipsy a křivky. V mém programu je pro vektorizaci použita pouze křivka a to proto, že křivkou respektive křivkami lze nahradit veškeré rastrové prvky. Křivka je tedy velice univerzální, mohu ji použít jak k nahrazení různých oblých tvarů, tak i rovných úseků. Ovšem zase na druhou stranu křivky nepatří k nejjednodušším matematickým prvkům, tudíž i vektorizace pomocí křivek bude výpočetně náročnější.

V následujícím textu této kapitoly se budu zabývat nahrazením segmentu křivkou. Je zde ale ještě jeden problém, a sice ten, že segment je liniového typu, ale může být různé délky a co hůře i různého tvaru. Z toho plyne, že segment může být všelijak zakřivený, může mít výrazná zalomení. A takový tvar nepůjde jakoukoliv křivkou nahradit tak, aby se výsledná křivka dostatečně podobala vstupnímu segmentu. Proto přeci jen ještě budou potřeba nějaké přípravné operace segmentu. Těmi operacemi bude rozdělení segmentů na více menších segmentů podle způsobu zakřivení, to znamená rozdělení na takové segmenty, které už půjde dosti dobře křivkou nahradit.

Po poslední přípravné operaci už se budou samostatné segmenty, vzniklé rozdělením, aproximovat křivkou. Pro aproximaci vybereme vhodný druh křivky a vyřešíme jak onu křivku na segment “napasovat“. Výsledek aproximace, tedy matematický tvar křivky potřebujeme vykreslit. To se provádí metodu zvanou rasterizace, což je opak k vektorizaci. Rasterizace vypočítává body ležící na křivce, respektive jejich souřadnice. Rasterizací je celý postup dovršen, máme již výkreslený vektorový výstup. Jediné co chybí je přiřadit jednotlivým vektorovým prvků tloušťku, popřípadě jas. Přesněji řečeno tloušťku přiřazujeme jen tenkým prvkům, poněvadž u tlustých prvků pracujeme s obrysem a ten je pak vyplněn určitou hodnotou jasu, což jsme probrali v předchozí kapitole.

5.1 Rozdělení segmentu podle změny směru

Vstupní segment již splňuje podmínky pro jeho aproximaci křivkou. Základní podmínkou je, aby segment byl liniové typu. Takový segment tedy vždy půjde jednou křivkou nahradit. Akorát že

zvolená křivka má vždy tvarové omezení, to podle typu zvolené křivky. Zato vstupní segment, i když je čarou o tloušťce jedna, může mít tvar v podstatě libovolný. Kdybychom takový segment, který je tvarově velmi složitý nahradili křivkou, tak aproximační metoda se sice pokusí nalézt tvar křivky co nejlépe odpovídající vstupnímu segmentu, ale je omezena vlastnostmi křivky. Tudíž dojde ke značnému zjednodušení, zdeformování výsledné křivky vůči vstupnímu segmentu. Proto potřebujeme tento segment rozdělit na segmenty, které už dokáže zvolená křivka tvarově dostatečně přesně representovat. Segment samozřejmě může být i tvarově jednoduchý, třeba rovný úsek, který půjde nahradit jednou křivkou přesně.

Potřebujeme získat informaci o tvaru, abychom věděli, zda-li segment rozdělit a kde. Budeme analyzovat tvar segmentu pomocí výpočtu gradientu [1], přesněji řečeno výpočtu směru gradientu. Vypočteme směr gradientu v každém pixelu segmentu. Známe již tedy průběh směru segmentu a v něm budeme hledat body, v nichž se segment rozdělí. Budou to body, ve kterých dochází k nějakým určitým změnám směru segmentu.

5.1.1 Výpočet gradientu

Gradient slouží pro nalezení hran v obraze. Přičemž obrazová hrana je dána vlastnostmi obrazového elementu a jeho okolím. Hrana je určena tím, jak prudce se mění hodnota obrazové funkce $f(x, y)$. A tuto změnu obrazové funkce udává její gradient. Vektorová veličina ∇ udává směr největšího růstu obrazové funkce $f(x, y)$, tedy směr gradientu ψ a strmost největšího růstu, tedy velikost (modul) gradientu $|\nabla f(x, y)|$. Hrana je tvořena pixely s velkým modulem gradientu. Výpočet modulu a směru gradientu je následující:

$$|\nabla f(x, y)| = \sqrt{f_x{}^2 + f_y{}^2}$$

$$\psi = \arctan\left(\frac{f_x}{f_y}\right)$$

Směr gradientu ψ je v radiánech a udává úhel mezi souřadnou osou x a radiusvektorem k bodu (x, y).

V mém případě vypočítávám gradient pro každý bod segmentu. Segment obsahuje pouze souřadnice bodu, a ne jeho hodnotu jasu, ta totiž pro nás není důležitá, protože má binární hodnoty. Proto je pro nás zbytečné počítat pro každý bod modul gradientu a počítáme jen jeho směr.

Pro výpočet jednotlivých složek vektoru gradientu použijeme konvoluci s jednoduchými maskami, které používají jen okolí 2x2 reprezentativního pixelu:

f_x :

-1	1
-1	1

f_y :

-1	-1
1	1

Okolí 2x2 je pro nás dostačující, protože gradient počítáme na jednotkové linii. Konvoluce s danou maskou se vypočítá následovně:

$$f_x = -1 \cdot (x, y) - 1 \cdot (x, y+1) + 1 \cdot (x+1, y) + 1 \cdot (x+1, y+1)$$

$$f_y = -1 \cdot (x, y) - 1 \cdot (x+1, y) + 1 \cdot (x, y+1) + 1 \cdot (x+1, y+1)$$

Směr gradientu se spočítá pro každý bod segmentu. Výsledné úhly v radiánech všech bodů segmentu budou uloženy v pomocném řetězci, se kterým budeme dále pracovat.

5.1.2 Detekce zalomení segmentu

První částí rozdělení segmentu je rozdělení segmentu podle zalomení. Bod zalomení segmentu je takové místo, kde dochází k náhlé a výrazné změně směru segmentu, která je trvalejšího rázu. To znamená, že zalomení musí být dost významné. Tím chceme potlačit malá zalomení, která jsou jen zubatostí segmentu zapříčiněné šumem obrazu nebo skeletonizační metodou.

Zalomení samozřejmě nelze vyjádřit jednou křivkou, proto musíme segment v bodě zalomení rozdělit a zalomení pak representovat dvěma křivkami, které se budou v místě zlomu dotýkat a budou spolu svírat úhel pod jakým je zalomen původní segment.

Vstupní segment je tvořen pixely a ke každému pixelu je přiřazen jeho směr gradientu. Směr gradientu je udávaný v radiánech a je uložený jako reálné číslo. Procházíme po sobě vždy dvě směrové hodnoty sousedních pixelů a v případě že se tyto hodnoty liší o danou mez, tak se jedná o eventuální zlomový bod. Jelikož ale chceme potlačit nevýznamná zalomení, procházíme i okolí těchto dvou bodů a i ta musí mít od sebe taktéž odlišné úhly o určitou mez. Znamená to, že určitý počet bodů před nalezenou dvojicí musí mít podobné úhly jako první bod s této dvojice a stejně tak určitý počet bodů za nalezenou dvojicí musí mít podobné úhly jako druhý bod z této dvojice. Zlom bude tam, kde spolu sousedí dvě skupiny pixelů s odlišnými úhly. Pak už se jen segment rozdělí v místě zlomu a tím je operace dokončena.

5.1.3 Detekce zakřivení segmentu

Prochází se segment a s ním i směry gradientů pro jednotlivé body. Při dosažení určitého stupně zakřivení, detekovaného podle následující metody, se již prošlá část segmentu prohlásí za samostatný segment a pokračuje se stejným způsobem v procházení zbytku segmentu.

Máme řetězec směrových úhlů a ty máme provázané s jednotlivými body segmentu. Budeme opět procházet řetězec směrů gradientů jednotlivých bodů. Pro lepší práci s ním tento řetězec hodnot zprůměrujeme s nějakým okolím. Tím se řetězec úhlů vyhladí a potlačí se nevýznamné hodnoty.

Zakřivení segmentu budeme detekovat jako místa, ve kterých dochází ke změně směru segmentu. Jsou to tedy místa, kde je nulová derivace, místa, kde není žádná změna směru (viz obrázek 6). Můžeme si takové body představit jako body překlopení segmentu. V řetězci úhlů tyto

body odpovídají lokálním maximům nebo lokálním minimům. V řetězci úhlů ponecháme jen tyto hodnoty. Jelikož jsou provázány s body segmentu, víme, o které konkrétní body se jedná. Mohlo by se zdát, že nalezené body už jsou hledanými body změny zakřivení, ale mezi body jsou i nevýznamná lokální maxima a minima. My ale potřebujeme najít body, kde dochází k výrazné změně směru segmentu. To provedeme vyloučením těchto nevýznamných hodnot z řetězce úhlů. A to tak, že při procházení řetězce úhlů hledáme shluky hodnot v určitém malém rozsahu a z tohoto shluku ponecháme jen nejvýznamnější hodnotu ať už lokální maximum nebo minimum. Teď už jsou v řetězci jen významné hodnoty, akorát vyloučením některých hodnot mohlo nastat, že v obraze nejsou jen lokální maxima a minima ale i hodnoty mezi nimi. Ty se jednoduše odstraní, opětovným použitím metody pro nalezení lokálních maxim a minim. Dosáhli jsme toho, že řetězec obsahuje jen hodny, které reprezentují významná lokální maxima a minima. Jak jsem již zmínil, hodnoty úhlů jsou provázány s daným bodem v segmentu. Takže nebude problém najít tyto body v segmentu a segment podle nich rozdělit.

5.2 Aproximace segmentu křivkou

Aproximaci segmentu můžeme chápat jako nahrazení popřípadě zjednodušení rastrového segmentu matematicky pospaným prvkem, tedy křivkou. Křivek je ovšem mnoho druhů a musíme vybrat takovou křivku, která je co možno nejjednoduššího typu a co nejlépe vyhovuje našemu použití. Po zvolení vhodné křivky hledáme takový její tvar, který co nejvíce odpovídá rastrovému segmentu. K hledání nejvhodnějšího tvaru bude použita metoda nejmenších čtverců, která nám zajistí výběr takového možného tvaru křivky, který se od originálního segmentu liší co nejméně, proto se této metodě také říká metoda nejmenší chyby.

5.2.1 Přehled křivek

Seznámíme se s principem křivek, s jednotlivými typy křivek s jejich vlastnostmi a vybereme typ křivky vhodný pro náš algoritmus. Všechny další informace o křivkách se dočtete v [3].

V reálném světě se často vyskytují tvary, které neodpovídají žádným základním geometrickým prvků, jako jsou přímky a kružnice, málo co je tak ideálně rovné nebo kulaté, aby se to dalo popsat těmito prvky. Proto potřebujeme něco, co umí popsat různě křivé tvary.

Vstupními daty křivky mohou být data generovaná přímo počítačem nebo data zadávána z vnějšku. Křivky jsou nejlépe reprezentovány funkcemi, ale funkce je problematické zadávat. Hledají se tedy metody, jak jednoduše zadat tvar nějaké křivky. Většinou se zadává jen několik řídících bodů a matematický aparát se o vygenerování křivky postará sám. První takovýto matematický model křivek a ploch, jenž umožňuje uživateli snadno je zadávat, začal používat v roce

1959 P. de Casteljau u firmy Citröen. V roce 1961 podobný systém používal P. Béziere u firmy Renault.

Použití křivek je naprosto neomezené, dá se ale začlenit do tří skupin, jako je definice fontů, definice objektů a určování různých trajektorií.

Modelování křivek probíhá na základě zadávání několika řídících bodů a matematický aparát z jejich polohy určí průběh křivky. Křivky se rozdělují na dvě základní skupiny a to právě podle způsobu zadávání řídících bodů:

1. Interpolační křivky

Generovaná křivka prochází řídícími bod.

Lagrangeova interpolační křivka: Jedna z nejstarších metod, jak vést křivku libovolným počtem řídících bodů. Pro výpočet polynomu je nutné znát souřadnice řídících bodů. Dále musí být zachována podmínka, že posloupnost interpolovaných bodů musí být seřazena podle x-ové souřadnice.

Fergusonovy kubiky: Je to metoda pro generování křivek, která je řízena dvěma koncovými body a jejich vektory. Jedná se o nejčastěji používanou interpolační křivku. Při změně pozice jednoho řídícího bodu dochází k nelokální změně tvaru křivky.

Catmull-Rom splajny: Vznikly nahrazením kontrolních bodů v matematickém vyjádření B-splajn křivky parametrickou funkcí t. Tyto křivky jsou používány především pro definování dráhy objektů v počítačové animaci. Křivka je definována řídícími body $P_0, P_1, ..., P_n$, ale vychází z bodu P_1 a končí v bodu P_{n-1}, to znamená, že neprochází prvním a posledním bodem. Nevýhodou těchto křivek je, že výsledný splajn obecně neleží v konvexní obálce svých řídících bodů.

2. Aproximační křivky

Generovaná křivka je řídícími body vhodně řízena a neprochází nimi.

Bézierovy kubiky: Vychází z Fergusonových kubik, kde použití řídících vektorů není příliš názorné, Bézierovy kubiky oproti tomu používají pouze řídícího polygonu bodů, který určuje tvar výsledné křivky. Při změně pozice jednoho bodu dochází pouze k lokální změně tvaru. Je tedy

invariantní k lokálním transformacím. Bézierova kubika je řízena čtyřmi body $P_0,...,P_3$, při čemž vychází z bodu P_0 a končí v bodu P_3, zbylé dva body určují její vyklenutí.

Obecné Bézierovy křivky: Jsou zobecněním Bézierových kubik. Křivka n-tého řádu vznikne z $n+1$ bodů řídícího polygonu $P_0, P_1,..., P_n$.

Racionální Bézierovy křivky: Jsou zobecněním Bézierových křivek. Každému bodu řídícího polygonu je přiřazeno reálné číslo, jehož změnou se mění tvar křivky. Díky tomu se pro změnu tvaru křivky nemusí měnit pozice bodů polygonu.

Coonsovy kubiky: Tato metoda má své silné použití díky dobrým geometrickým vlastnostem hlavně v navrhování ploch. Coonsova kubika se zadává čtyřmi řídícími body $P_0,...,P_3$, ale konce křivky nevychází z řídících bodů nýbrž z antitěžiště trojúhelníků P_0, P_1, P_2 a P_1, P_2, P_3.

Coonsův B-splajn: Říká se mu také uniformní neracionální kubický B-splajn. Vzniká skládáním Coonsových kubik, a to tak, že výslednou křivku, tvořenou polygonem $P_0, P_1,..., P_n$, budeme skládat z Coonsových oblouků tím způsobem, že pro první oblouk použijeme body $P_0,..., P_3$ a pro další oblouk body $P_1,..., P_4$ atd. Výsledná křivka se nazývá B-splajn. Vlastností B-splajnů je, že má ve všech vnitřních bodech spojitost druhého řádu. Dále je výhodné, že změnou jednoho bodu dojde pouze k lokální změně čtyř oblouků, se kterými bod souvisí.

NURBS křivky: Celým názvem, neuniformní racionální B-splajn křivky. Neuniformita znamená, že vzdálenost uzlů, ve smyslu parametru t, nemusí být konstantní. Jsou dvojím zobecněním B-splajn křivek. Uzlový vektor pro křivku NURBS prochází prvním a posledním bodem řídicího polynomu.

Na základě vlastností popsaných v předchozím přehledu už můžeme vybrat nejvhodnější křivku pro naše použití.

Naprosto nepoužitelná je Lagrangeova interpolační křivka, jelikož její interpolační body musí být seřazeny postupně podle souřadnice x. Nevhodné je použití Catmull-Rom splajnů, především proto, že výsledný splajn neleží v konvexní obálce řídících bodů. Dále vyloučíme Coonsovy kubiky, protože konce křivky nevychází z řídících bodů. Stejně tak i Coonsův B-splajn, který též nevychází z řídících bodů. Jeho použití je navíc zbytečné, protože se jedná složení z více Coonsových kubik a náš segment, který chceme křivkou aproximovat je již nahraditelný jednou kubikou. Zbytečné je i ze

stejného důvodu použití křivky NURBS i přesto, že prochází prvním a posledním bodem řídícího polynomu.

Zbývají nám už jen Fergusonovy kubiky a různé typy Bézierových křivek. Všechny tyto metody už je v podstatě možné použít. Takže je potřeba vybrat metodu, která je pro nás nejefektivnější. Fergusonovy a Bézierovy kubiky jsou velice podobné. Radši ale zvolíme Bézierovy kubiky, protože mají lepší způsob řízení tvaru a celkově jsou rozšířenější. Použití obecných Bézierových křivek a racionálních Bézierových křivek je také možné, ale nebude tolik efektivní, z důvodů jako použití splajnů, tedy že segment se dá aproximovat pouhou kubikou.

Pro aproximaci jsem vybral Bézierovy kubiky, jejich podrobnější popis včetně matematického popisu bude v následující kapitole.

5.2.2 Aproximace Bézierovou kubikou

Budeme se zabývat algoritmem na aproximaci vstupního segmentu Bézierovou kubikou navrženém pro vektorizaci fontů v [4], akorát s tou změnou, že má aproximační metoda pracuje s pevně danými koncovými řídícími body. Pomocí metody nejmenších čtverců [6] nalezneme takový tvar kubiky, který nejlépe odpovídá segmentu.

Vstupní segment, který chceme aproximovat, je dán množinou bodů $p_i = (x_i, y_i)$, pro $i = 1,...,n$. Parametrizovaná Bézierova 2D křivka $Q(t) = [x(t), y(t)]$ má tvar:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y, \qquad t \in <0,1>$$

Základní tvar křivky je $P(t) = P_0 B_0(t) + P_1 B_1(t) + P_2 B_2(t) + P_3 B_3(t) = \sum_{i=0}^{3} P_i B_i(t)$, kde $P_0,...,P_3$ jsou řídícími body křivky, o souřadnicích (x_i, y_i). $B_0,...,B_3$ jsou Bernsteinovy polynomy stupně 3:

$$B_0(t) = (1-t)^3$$
$$B_1(t) = 3t(1-t)^2$$
$$B_2(t) = 3t^2(1-t)$$
$$B_3(t) = t^3$$

Po dosazení do základního tvaru pro x a pro y dostaneme tvar:

$$P_x(t) = P_{0_x}(1-t)^3 + 3P_{1_x} t(1-t)^2 + 3P_{2_x} t^2(1-t) + P_{3_x} t^3$$
$$P_y(t) = P_{0_y}(1-t)^3 + 3P_{1_y} t(1-t)^2 + 3P_{2_y} t^2(1-t) + P_{3_y} t^3$$

Pro určení koeficientů založených na daných bodech segmentu je hodnota parametru t_i pro každý bod segmentu vypočítána jako:

$$t_i = \begin{cases} 0, & i = 1 \\ \dfrac{\textit{délka segmentu od } p_1 \textit{ po } p_i}{\textit{celková délka segmentu od } p_1 \textit{ do } p_n}, & 1 < i \le n \end{cases}$$

Poté suma čtverců získaných ze vzdálenosti mezi body segmentu p_i a body na křivce v t_i je vypočítána jako:

$$S = \sum_{i=1}^{n} \left[\textit{vzdálenost mezi } P(t_i) \textit{ a } p_i \right]^2$$

Konkrétně:

$$S(P_0, P_1, P_2, P_3) = \sum_{i=1}^{n} \left(P_{0_x}(1-t)^3 + 3P_{1_x} t(1-t)^2 + 3P_{2_x} t^2(1-t) + P_{3_x} t^3 - x_i \right)^2 +$$
$$\sum_{i=1}^{n} \left(P_{0_y}(1-t)^3 + 3P_{1_y} t(1-t)^2 + 3P_{2_y} t^2(1-t) + P_{3_y} t^3 - y_i \right)^2$$

Pomocí parciálních derivací podle P_0, P_1, P_2, P_3 budeme hledat stacionární body, tedy body, ve kterých nastává lokální extrém cenové funkce S.

$$\frac{\partial S}{\partial P_0} = 2\sum_{i=1}^{n} \left(P_{0_x}(1-t)^3 + 3P_{1_x} t(1-t)^2 + 3P_{2_x} t^2(1-t) + P_{3_x} t^3 - x_i \right)(1-t)^3 =$$
$$= 2\left(P_{0_x} \sum_{i=1}^{n} (1-t)^6 + 3P_{1_x} \sum_{i=1}^{n} t(1-t)^5 + 3P_{2_x} \sum_{i=1}^{n} t^2(1-t)^4 + P_{3_x} \sum_{i=1}^{n} t^3(1-t)^3 - \sum_{i=1}^{n} x_i(1-t)^3 \right)$$

$$\frac{\partial S}{\partial P_1} = 2\sum_{i=1}^{n} \left(P_{0_x}(1-t)^3 + 3P_{1_x} t(1-t)^2 + 3P_{2_x} t^2(1-t) + P_{3_x} t^3 - x_i \right) t(1-t)^2 =$$
$$= 2\left(P_{0_x} \sum_{i=1}^{n} t(1-t)^5 + 3P_{1_x} \sum_{i=1}^{n} t^2(1-t)^4 + 3P_{2_x} \sum_{i=1}^{n} t^3(1-t)^3 + P_{3_x} \sum_{i=1}^{n} t^4(1-t)^2 - \sum_{i=1}^{n} x_i t(1-t)^2 \right)$$

$$\frac{\partial S}{\partial P_2} = 2\sum_{i=1}^{n} \left(P_{0_x}(1-t)^3 + 3P_{1_x} t(1-t)^2 + 3P_{2_x} t^2(1-t) + P_{3_x} t^3 - x_i \right) t^2(1-t) =$$
$$= 2\left(P_{0_x} \sum_{i=1}^{n} t^2(1-t)^4 + 3P_{1_x} \sum_{i=1}^{n} t^3(1-t)^3 + 3P_{2_x} \sum_{i=1}^{n} t^4(1-t)^2 + P_{3_x} \sum_{i=1}^{n} t^5(1-t) - \sum_{i=1}^{n} x_i t^2(1-t) \right)$$

$$\frac{\partial S}{\partial P_3} = 2\sum_{i=1}^{n} \left(P_{0_x}(1-t)^3 + 3P_{1_x} t(1-t)^2 + 3P_{2_x} t^2(1-t) + P_{3_x} t^3 - x_i \right) t^3 =$$
$$= 2\left(P_{0_x} \sum_{i=1}^{n} t^3(1-t)^3 + 3P_{1_x} \sum_{i=1}^{n} t^4(1-t)^2 + 3P_{2_x} \sum_{i=1}^{n} t^5(1-t) + P_{3_x} \sum_{i=1}^{n} t^6 - \sum_{i=1}^{n} x_i t^3 \right)$$

Výsledné parciální derivace položíme rovny nule, tím můžeme získat takové body, které vytvoří minimální cenovou funkci S. A po dalších úpravách dosáhneme tvaru:

$$P_{0_x}\sum_{i=1}^{n}(1-t)^6+3P_{1_x}\sum_{i=1}^{n}t\,(1-t)^5+3P_{2_x}\sum_{i=1}^{n}t^2(1-t)^4+P_{3_x}\sum_{i=1}^{n}t^3(1-t)^3=\sum_{i=1}^{n}x_i\quad(1-t)^3$$

$$P_{0_x}\sum_{i=1}^{n}t\,(1-t)^5+3P_{1_x}\sum_{i=1}^{n}t^2(1-t)^4+3P_{2_x}\sum_{i=1}^{n}t^3(1-t)^3+P_{3_x}\sum_{i=1}^{n}t^4(1-t)^2=\sum_{i=1}^{n}x_i t\quad(1-t)^2$$

$$P_{0_x}\sum_{i=1}^{n}t^2(1-t)^4+3P_{1_x}\sum_{i=1}^{n}t^3(1-t)^3+3P_{2_x}\sum_{i=1}^{n}t^4(1-t)^2+P_{3_x}\sum_{i=1}^{n}t^5(1-t)\quad=\sum_{i=1}^{n}x_i t^2(1-t)$$

$$P_{0_x}\sum_{i=1}^{n}t^3(1-t)^3+3P_{1_x}\sum_{i=1}^{n}t^4(1-t)^2+3P_{2_x}\sum_{i=1}^{n}t^5(1-t)\quad+P_{3_x}\sum_{i=1}^{n}t^6\qquad=\sum_{i=1}^{n}x_i t^3$$

Stejným způsobem získáme rovnice pro souřadnice y. Takto vzniklá soustava o čtyřech rovnicích by pro nás byla výhodná, pokud bychom chtěli počítat všechny čtyři řídící body P_0, P_1, P_2, P_3, tedy pro případ, že bychom neměli pevně definovány koncové body křivky. Měli bychom soustavu o čtyřech rovnicích a o čtyřech neznámých. Pomocí Gaussovy eliminace by se dali tyto neznámé vyjádřit.

Jenže my máme koncové řídící body P_0 a P_3 pevně dány a to prvním a posledním bodem segmentu, jedná se o segmentové body p_1 a p_n. Máme tedy jen dvě neznámé a to řídící body P_1 a P_2. K jejich vyjádření nám budou stačit jen dvě rovnice, použijeme z naší soustavy například první dvě:

$$p_{1_x}\sum_{i=1}^{n}(1-t)^6+3P_{1_x}\sum_{i=1}^{n}t\,(1-t)^5+3P_{2_x}\sum_{i=1}^{n}t^2(1-t)^4+P_{n_x}\sum_{i=1}^{n}t^3(1-t)^3=\sum_{i=1}^{n}x_i\quad(1-t)^3$$

$$p_{1_x}\sum_{i=1}^{n}t\,(1-t)^5+3P_{1_x}\sum_{i=1}^{n}t^2(1-t)^4+3P_{2_x}\sum_{i=1}^{n}t^3(1-t)^3+P_{n_x}\sum_{i=1}^{n}t^4(1-t)^2=\sum_{i=1}^{n}x_i t\quad(1-t)^2$$

Již máme dvě rovnice o dvou neznámých, po vypočítání jednotlivých sum vyjádříme neznámé P_1 a P_2, tedy řídící body křivky, která má nahradit vstupní segment.

5.3 Výsledné vykreslení

Každý segment máme nyní nahrazený čtyřmi řídícími body Bézierovy křivky respektive kubiky. Přičemž první a poslední z těchto řídících bodů je krajním bodem segmentu. Krajní body segmentu ponecháváme z důvodů zachovaní souvislostí mezi jednotlivými segmenty. Zbylé dva řídící body jsou vypočítány tak, že křivka jimi řízená co nejlépe tvarově odpovídá rastrovému segmentu.

Známe tedy funkci křivky a její parametry pro každý segment, ale potřebujeme mít také nějaký viditelný výstup, proto je křivky potřeba vykreslit. Vykreslení se provede rasterizační metodou, což je metoda, která na základě matematické funkce zjistí průběh křivky a nalézá po určitém kroku body ležící na této křivce a přiřadí jim rastrovou pozici. Tyto body se pak propojí úsečkami, které se taktéž vyrastrují. Tím je křivka vykreslena.

Máme výstupní obraz tvořen množinou vyrastrovaných křivek. Akorát, že takový výstup se příliš nepodobá originálnímu obrazu. Naším záměrem bylo, aby výsledný obraz se co nejvíce podobal vstupnímu obrazu. Potřebujeme přiřadit prvkům našeho obrazu tloušťku a barvu co nejvíce podobnou prvkům v originálním obraze.

Náš vykreslený obraz je v podstatě tvořen vektorizovanými skeletony a vektorizovanými obrysy. Skeletony jsou vyextrahované liniové prvky tenkých objektů a obrysy jsou vyextrahovanými liniovými prvky tlustých objektů. Z toho je docela patrné, že vektorizovaným skeletonům se dá při vykreslování přiřadit tloušťka a barva odpovídající původním prvkům. U vektorizovaných obrysů nám přiřazení tloušťky nebude k ničemu. Jedná se o obrysy tlustých prvků a tyto obrysy je třeba vyplnit barvou získanou z originálu. Metodu vyplňování oblastí jsme už probrali v kapitole 4.4.

5.3.1 Algoritmus De Casteljau

Používá se pro rasterizaci Bézierových kubik, tedy výpočtu bodů ležících na Bézierove kubice, viz literatura [3], [7]. Pro výpočet využívá rekurzivní definice Bernsteinova polynomu:

$P_{i,j}(t) = (1-t) \cdot P_{j-1,i-1} + t \cdot P_{j,i-1}$, kde $i = 1,2,...,n$; $j = i, i+1,...,n$ a parametr t určuje poměr dělení stran řídícího polygonu.

Výpočet spočívá v generování křivky se zvětšujícím se parametrem t. Velikost kroku parametru t určuje přesnost, respektive počet vypočítaných bodů Bézierovy kubiky. V mém algoritmu je velikost kroku definována na základě délky rastrového segmentu.

Vstupními daty algoritmu jsou řídící body Bézierovy kubiky. Výstupem je řetězec bodů ležících na křivce a tyto body se následně propojí úsečkami a tím vykreslí aproximovanou křivku.

5.3.2 Rekonstrukce jasu a tloušťky objektů

Ve vstupním obraze je každý prvek zobrazen určitou barvou/jasem a rozlohou/tloušťkou. Ale výstupní vykreslený obraz je tvořen pouze liniovými prvky. Detekce a rekonstrukce tloušťky a jasu se bude provádět dvěma způsoby.

První způsob se bude aplikovat na linie, které vznikly extrahováním (skeletonizací) tenkých prvků. Takové linie, když se vykreslí odpovídající tloušťkou, tak se budou dostatečně podobat originálu. Samotná detekce tloušťky a jasu se bude provádět už pro rastrový segment. Tloušťka se pro celý segment bude detekovat načítáním hodnot vzdálenostní mapy vstupního obrazu podle souřadnic bodů v segmentu. Z hodnot získáme průměr a máme informaci o tloušťce pro daný segment. S jasem

to bude podobné, akorát, že se nebude procházet vzdálenostní mapa, ale originální obraz. Křivka se určitou tloušťkou a barvou vykreslí pomocí rasterizace úsečkami. Pro kreslení úseček existuje v OpenCV funkce CvLine, která umí vykreslovat barvu i tloušťku.

Druhý způsob se bude aplikovat na linie, které vznikly extrahování (nalezením obrysů) tlustých prvků. Nemá smysl obrysové linie vykreslovat zadanou tloušťkou, tyto linie vytvoří uzavřený polygon a ten budeme vyplňovat (popsáno v kapitole 4.4) nadetekovaným jasem. Jasovou hodnotu pro tlusté prvky zjišťujeme již v době zpracování tlustých prvků. Tam projdeme každý pixel tohoto prvku a načítáme jeho jasové hodnoty v originálním obraze. Hodnoty se zprůměrují a máme jasovou informaci pro daný objekt, kterou neseme až do vyplňování polygonu.

6 Závěr

Cílem mé bakalářské práce bylo prostudovat dostupnou literaturu, týkající se problematiky vektorizace a na základě získaných informací vybrat metody a odůvodnit jejich použití. Dle zvolených metod navrhnout možné řešení vektorizace rastrového obrazu a toto řešení naimplementovat. Výsledné řešení pak na základě testovacích dat analyzovat a shrnout výhody a nevýhody obsažených metod a celkového postupu.

V první části mé práce jsem popisoval předzpracování obrazu před samotnou vektorizací. Předzpracování obrazu je extrahování takových prvků z obrazu, které už lze vektorizovat. Jako první krok předzpracování jsem popsal rozdělení obrazu na dva obrazy podle tloušťky (kapitola 2). Metoda obsahovala rozložení obrazu do jasových skupin a v každé skupině rozdělování podle detekované tloušťky. Tato část předzpracování je ovšem závislá na typu vstupního obrazu a na jeho šumu. Tento způsob rozdělení obrazu na objekty podle tloušťky počítá s tím, že jednotlivé prvky v obraze jsou representovány určitou barvou. Proto mé testování bylo zaměřeno především na mapy, kde každý prvek mapy je representován veskrze uniformní barvou. Šum obrazu jsem musel vyřešit ručním vyhlazováním obrazu, protože každá mapa může být různě zašuměná, to se odvíjí od způsobu získání mapy (skenování, fotografování, vytvořená v PC).

V dalších kapitolách jsem se věnoval nalezením liniových prvků v obou obrazech (kapitoly 3 a 4). U prvního obrazu s méně tlustými prvky jsem hledal středové linie metodou skeletonizace. U této metody dochází k různým roztřepením, což jsem částečně vyřešil odstraněním krátkých větví. U druhého obrazu s tlustšími prvky jsem hledal obrysové linie a přitom jsem se zmínil i o hranových detektorech, které by šly taktéž použít.

V druhé části práce jsem se zaměřil na samotnou vektorizaci (kapitola 5). Výsledkem první popsané části byl liniový prvek vyextrahovaný z původního obrazu, který již lze vektorizovat. Zabýval jsem se výběrem geometrického prvku, kterým se bude liniový prvek vektorizovat (aproximovat). Zvolil jsem právě křivku z důvodu její univerzálnosti. Existuje řada způsobů, jak liniový prvek aproximovat křivkou, většina z nich vychází z metody nejmenších čtverců. Libovolný liniový prvek aproximují jednou křivkou, a pokud není dostatečná podobnost s originálem, přidávají další křivky. Zvolil jsem odlišný postup, liniový prvek vždy nahrazuji jen jednou křivkou, ale ještě před tím tuto linii rozděluji na takové části, které lze jednou křivkou nahradit. Tímto dojde k zjednodušení samotného aproximačního procesu.

Hlavním přínosem mé práce je seskupení různých technik, které všechny dohromady vytváří proces vektorizace. Vektorizace je velmi složitá procedura, ale podařilo se mi s jistými odchylkami vektorově prezentovat vstupní obraz. Ovšem ještě nelze dosáhnout odpovídajícího výstupu bez zásahu uživatele, který musí nastavovat potřebné parametry. Nevýhodou nebo spíše vlastností mnou navrženého postupu je, že přijatelně zpracuje pouze takový vstupní obraz, kde jsou prvky

jednoznačně barevně identifikovány. Tento postup je tedy navržen především pro mapy. Celkovou nevýhodou programu je, že dochází ke zkreslení tenkých prvků vůči vstupnímu obrazu. Je to způsobeno metodou skeletonizace, ale také nepřesnou detekcí tloušťky, která se projeví právě u tenkých prvků.

Vektorizace je vskutku velmi složitou a rozsáhlou problematikou. Zcela jistě je tedy možná řada vylepšení a rozšíření mého programu. Velice žádané by bylo zpracování obrazu bez zásahu uživatele. Tedy, že by si systém sám měnil parametry podle vstupního obrazu. Jako další se naskýtá rozšíření vektorizačního procesu na libovolný vstup. Ne tedy jen na mapy, musel by se ale upravit způsob detekce prvků v obraze, například za použití hranových detektorů. Určitě by stálo za vyzkoušení nahradit metodu zpracování tenkých prvků, kterou je skeletonizace, jelikož tato metoda způsobuje značné deformace. Důvodem proč se ale vektorizace provádí, je především v různých operacích, které vektorový formát umožňuje. Proto bych doporučoval program o takovéto operace, jako je zvětšování zmenšování, rozšířit. S tím samozřejmě zavést i možnost ukládání ve vektorovém formátu.

Literatura

[1] Hlaváč, V., Sedláček, M.: Zpracování signálu a obrazu. Elektrotechnická fakulta ČVUT v Praze, 1999.

[2] Altman, P.: Digitalizace mapy. Diplomová práce, Praha, Univerzita Karlova, Matematicko-fyzikální fakulta, 2004. Dokument je dostupný na URL http://pal.altmanoptik.com/download/DigitalizaceMapy.pdf (duben 2007).

[3] Alexandr, L.: Výuka počítačové grafiky cestou WWW. Diplomová práce, Vysoké učení technické v Brně, Fakulta elektrotechniky a informatiky. Dokument je dostupný na URL http://lubovo.misto.cz/_MAIL_/curves/obsah.html (duben 2007).

[4] Itoh, k., Ohno, Y.: A Curve Fitting Algorithm for Character Fonts. Yokohama, Keio University, Faculty of Sciences and Technology, 1993.

[5] Open Source Computer Vision Library. Manuál, Intel Corporation, 1999 – 2001. Dokument je dostupný na URL http://opencvlibrary.sourceforge.net/ (květen 2007).

[6] Mařík, R.: Metoda nejmenších čtverců. Mendelova zemědělská a lesnická univerzita v Brně, 2006. Dokument je dostupný na URL http://old.mendelu.cz/~marik/prez/mnc-cz.pdf (duben 2007).

[7] Žára, J., Beneš, B., Felkel, P.: Moderní počítačová grafika, 1, Computer Press Praha, 1998.

Seznam příloh

Příloha 1. Programová dokumentace

Čtenáře zde seznámím s možnostmi programu a jeho ovládáním. Způsob seznámení bude ve formě tutoriálu, kde předvedu postupně cely proces převodu obrazu.

Obsah CD

Dodané CD obsahuje elektronickou verzi písemné zprávy, zdrojové kódy a spustitelný program včetně vzorových vstupních a výstupních obrázků.

/Bakalářská práce/Curve Fitting - spustitelný program Curve_fitting.exe
/Bakalářská práce/Curve Fitting /Sources – zdrojové kódy
/Bakalářská práce/Curve Fitting /Samples – vzorové obrázky
/Bakalářská práce/Písemná zpráva – Bakalářská práce.pdf

Spuštění programu

Program se neinstaluje, je spustitelný přímo z CD. Načítání vstupního obrazu se děje přímo při spuštění. Název obrazu se zadává jako parametr. Ale není povinnost parametr zadávat. Při spuštění bez parametrů se program pokusí načíst implicitně zadaný název souboru „Input.bmp“.

Zadávání parametrů: nazev_programu -input nazev_obrazu.*
Příklad spuštění: Curve_fitting –input Input.bmp

Program pracuje s šedotónovými obrazy ve formátu *.bmp nebo *.jpg.

Práce s programem

Po spuštění programu se objeví dvě okna, přičemž je aktivní pouze okno Input (viz obrázek 2), obsahuje načtený vstupní obraz a slouží pro úpravy obrazu před vektorizací. Okno Output je zatím neaktivní, je připraveno pro vektorizovaný výstup.

Ovládání programu se provádí následujícími klávesami (viz obrázek 1):

‘s‘ Pro ukládání aktivních obrazů, vstupní upravený obraz ukládá ve tvaru „ImageInput.bmp“, výsledný vektorový obraz ukládá ve tvaru „ImageOutput.bmp“, pokud není okno Output aktivní, uloží pouze „ImageInput.bmp“. Pokud je spuštěný proces vektorizace, ukládá i chybovou zprávu „error.txt“ obsahující rozdíl mezi upraveným vstupním obrazem a vektorizovaným výstupním obrazem.

‘p‘ Spustí vektorizační proces, v okně Output se objeví vektorizovaný obraz.

Následující příkazy pracují s vektorovým výstupem, tedy fungují až po spuštění vektorizačního procesu klávesou „p“.

‘m‘ Vyhladí vektorizovaný výstup Gaussem.

‘a‘ Při vykreslování vektorového výstupu použije antialiasing.

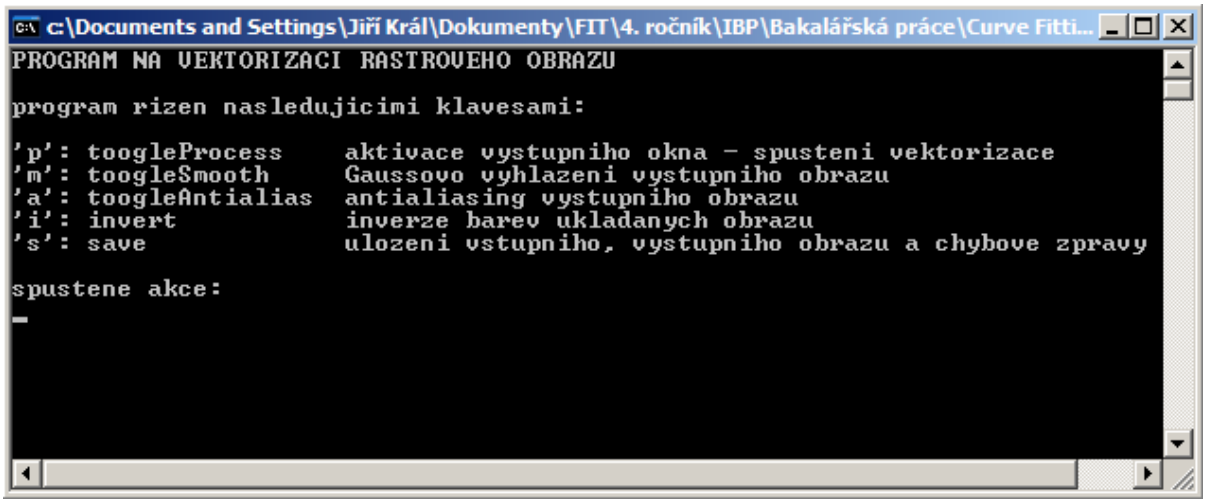
‘i‘ Slouží pro invertování barev ukládaných obrazů, je to z důvodu, že program pro svou práci vstupní obraz invertuje, tak aby pixely s hodnotou nula byly ty neaktivní.

Program je také řízen dvěma posuvnými tlačítky (viz obrázek 2):

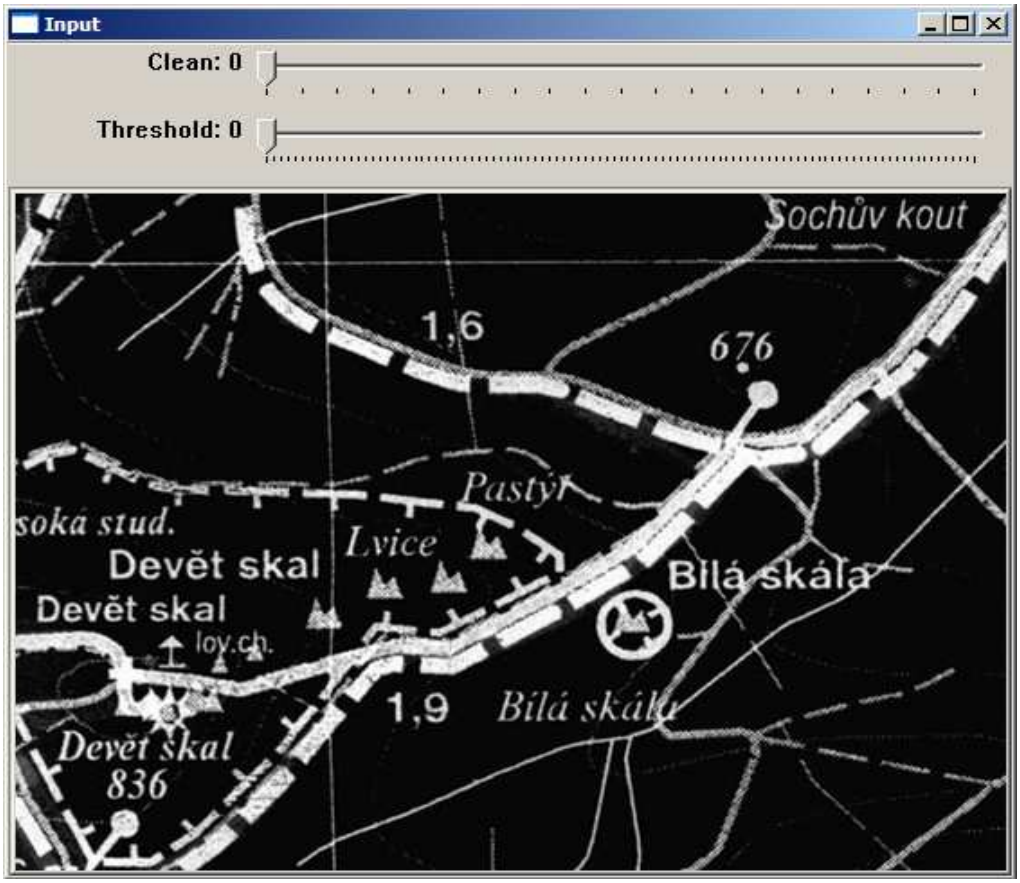
Threshold: Nedělá nic jiného, než že potlačuje nevýznamné, příliš tmavé hodnoty vstupního obrazu, zrychluje algoritmus. Hodnotu Threshold běžně volím mezi 30 a 60.

Clean: Provádí vyhlazení vstupního obrazu, homogenizaci jasových hodnot jednotlivých prvků v obraze. Potlačuje šum v obraze. Tím, že každému prvku přiřadí téměř konstantní hodnotu jasu, je následně snazší detekce prvků. Hodnota se obvykle nastavuje mezi 10 a 20. Na obrázku 3 je vidět, jak se vyhladily jednotlivé objekty obrazu při Clean = 20. Jednotlivé objekty už mají téměř konstantní hodnotu jasu.

Při takovýchto parametrech, jaké jsou nastaveny na obrázku 3 už můžeme spustit proces vektorizace (viz obrázek 4). Jedná se už o výsledný obraz, který se dá již uložit. Je ale vhodné jej ještě vyhladit (viz obrázek 5).

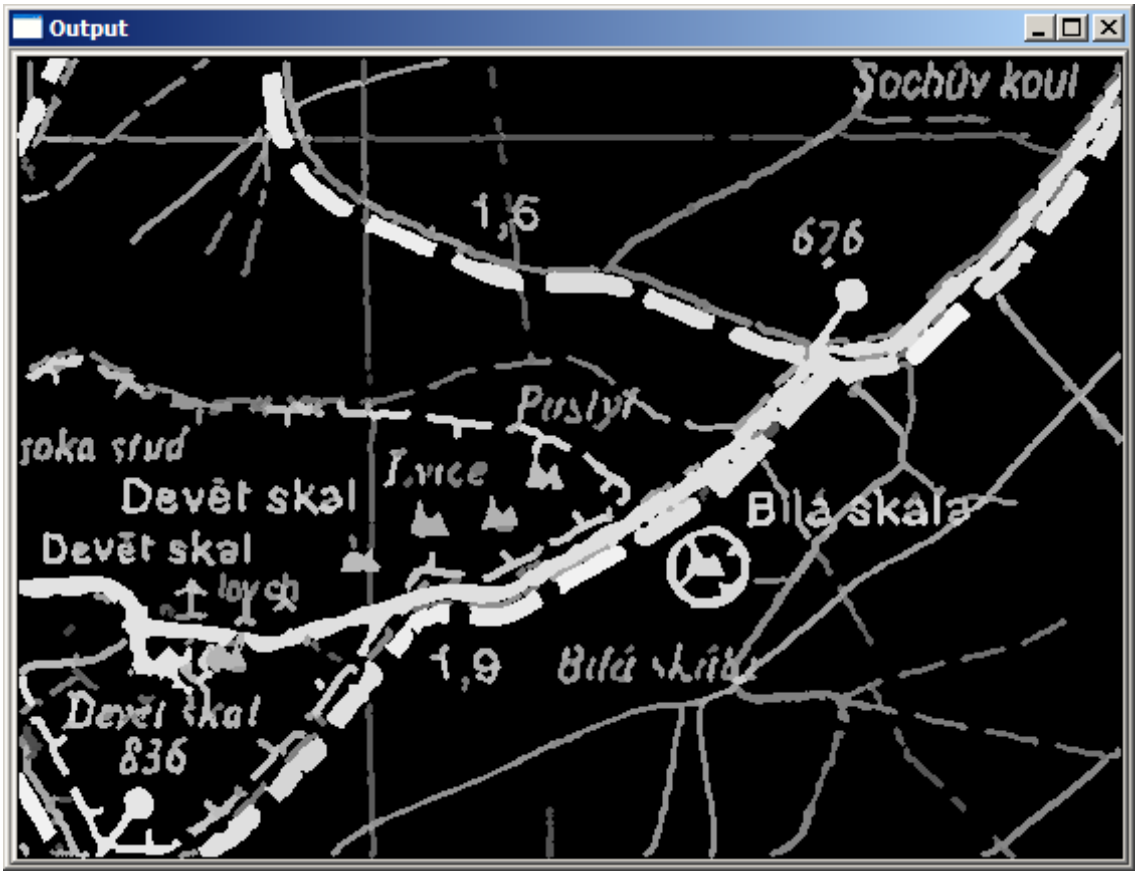


Obrázek 1: Klávesy pro ovládání programu.

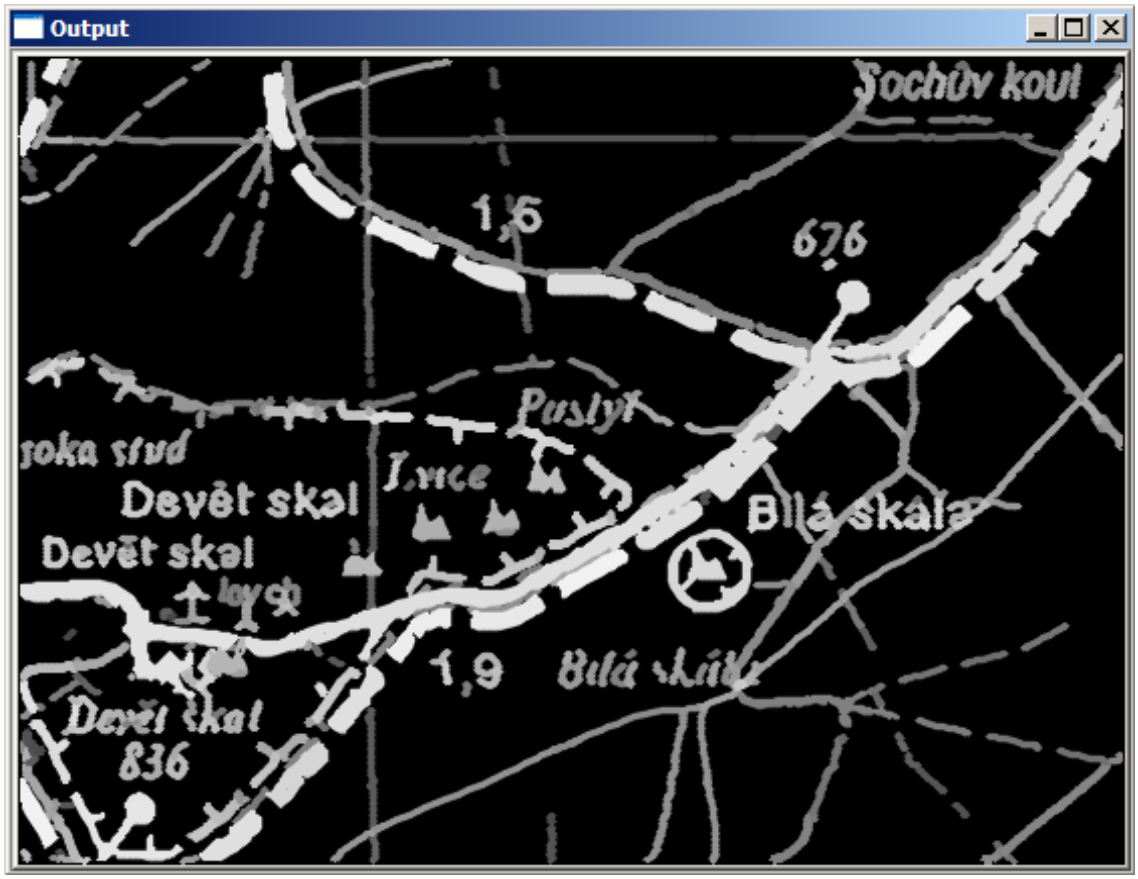


Obrázek 2: Okno pro předzpracování obrazu s načteným původním obrazem.

Obrázek 3: Vliv na obraz změnou hodnoty parametru Clean.



Obrázek 4: Vektorizovaný obraz z předchozího upraveného obrazu.



Obrázek 5: Vektorizovaný obraz vykreslený s antialiasingem.

Popis implementace

Všechny mé postupy popsané v celé práci jsou implementovány v programu CurveFitting. Implementace je provedena v jazyce C++ s využitím volně šiřitelné knihovny OpenCV. Knihovna je navržena pro zpracování obrazů a je zde řada obrazových funkcí, které ve svém algoritmu využívám. Následně uvedu popis nejdůležitějších funkcí programu:

- SeparateObjects: Rozdělí vstupní obraz na obrazy s dvěma typy objektů (s menší a větší tloušťkou).
- FindJoin: Ve vstupním obraze, kterým je třeba skeleton, detekuje koncové body a křižovatky.
- ConvertImage2Segment: Uloží do segmentu části skeletonu definované detekovanými body z funkce FindJoin.
- FindCorner: Na základě výpočtu gradientu detekuje v každém segmentu body zalomení a segment v takových bodech rozděluje.
- SeparateSegment: Opět na základě výpočtu gradientu rozděluje segment na segmenty aproximovatelné jednou Bézierovou kubikou.
- Raster2Vector: Aproximace segmentu Bézierovou kubikou pomocí metody nejmenších čtverců.
- DeCasteljau: Vstupem jsou řídící body Bézierovy kubiky, na základě kterých vypočítá body ležící na takto definované kubice.