SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Stavebná fakulta

Evidenčné číslo: SvF-104289-68425

Grafové algoritmy na hľadanie najkratších ciest v grafoch a ich aplikácia v spracovaní obrazu.

Diplomová práca

2016 **Bc. Richard Rožnovják**

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

Stavebná fakulta

Evidenčné číslo: SvF-104289-68425

Grafové algoritmy na hľadanie najkratších ciest v grafoch a ich aplikácia v spracovaní obrazu.

Diplomová práca

Študijný program: matematické a počítačové modelovanie
Študijný odbor: 9.1.9. aplikovaná matematika
Školiace pracovisko: Katedra matematiky a deskriptívnej geometrie
Vedúci záverečnej práce: Mgr. Mária Ždímalová, PhD.

Bratislava 2016 **Bc. Richard Rožnovják**

POKYNY
na vypracovanie diplomovej práce

Úvodné ustanovenie

V zmysle zákona č. 131/2002 Z. z. o vysokých školách a o zmene a doplnení niektorých zákonov v znení neskorších predpisov je súčasťou štúdia podľa každého študijného programu aj záverečná práca. Jej obhajoba patrí medzi štátne skúšky. Záverečnou prácou pri štúdiu podľa študijného programu druhého stupňa je diplomová práca. Podkladom na vypracovanie diplomovej práce je zadanie diplomovej práce

Štruktúra záverečnej práce

- titulný list,
- zadanie záverečnej práce,
- pokyny na vypracovanie,
- vyhlásenie autora,
- názov a abstrakt v slovenskom a v anglickom jazyku (spolu v rozsahu jednej strany),
- obsah s očíslovaním kapitol,
- zoznam príloh,
- zoznam skratiek a značiek,
- text samotnej práce (odporúčané členenie),
  - úvod,
  - súčasný stav problematiky,
  - ciele záverečnej práce,
  - vlastné riešenie členené na kapitoly podľa charakteru práce,
  - zhodnotenie dosiahnutých výsledkov resp. navrhnutých riešení,
  - záver,
- resumé v slovenskom jazyku v rozsahu spravidla 10 % rozsahu ZP (len pre práce vypracované v cudzom jazyku),
- zoznam použitej literatúry,
- prílohy (výkresy, tabuľky, mapy, náčrty) vrátane postera s rozmermi 1000x700 mm.

Rozsah a forma

1. Obsah a forma záverečnej práce musí byť spracovaná v zmysle vyhlášky MŠVVaŠ SR č. 233/2011 Z. z., ktorou sa vykonávajú niektoré ustanovenia zákona č. 131/2002 Z. z. a v zmysle Metodického usmernenia č. 56/2011 o náležitostiach záverečných prác.
2. Vyžadovaný rozsah diplomovej práce je 30 až 50 strán. Odovzdáva sa v dvoch vyhotoveniach. Jedno vyhotovenie musí byť viazané v pevnej väzbe (nie hrebeňovej) tak, aby sa jednotlivé listy nedali vyberať. Rozsiahle grafické prílohy možno v prípade súhlasu vedúceho práce odovzdať v jednom vyhotovení.

3. Autor práce je povinný vložiť prácu v elektronickej forme do akademického informačného systému. Autor zodpovedá za zhodu listinného aj elektronického vyhotovenia.
4. Po vložení záverečnej práce do informačného systému, predloží autor fakulte ním podpísaný návrh licenčnej zmluvy. Návrh licenčnej zmluvy je vytvorený akademickým informačným systémom.
5. Odporúčaný typ písma je Times New Roman, veľkosť 12 a je jednotný v celej práci. Odporúčané nastavenie strany - riadkovanie 1,5, okraj vnútorný 3,5 cm, vonkajší 2 cm, zhora a zdola 2,5 cm, orientácia na výšku, formát A4.
6. Obrázky a vzorce sa číslujú v rámci jednotlivých kapitol (napr. obr. 3.1 je obrázok č. 1 v kapitole 3). Vzorce sa číslujú na pravom okraji riadku v okrúhlych zátvorkách - napr. (3.1).
7. Všetky výpočty musia byť usporiadané tak, aby bolo možné preveriť ich správnosť.
8. Pri všetkých prevzatých vzorcoch, tabuľkách, citovaných častiach textu musí byť uvedený prameň.
9. Citovanie literatúry vrátane elektronických materiálov sa uvádza podľa STN ISO 690 (01 0197): 2012. *Informácie a dokumentácia. Návod na tvorbu bibliografických odkazov na informačné pramene a ich citovanie.*
10. Príklad zoznamu bibliografických odkazov:
    ABELOVIČ, J. a kol.: *Meranie v geodetických sieťach.* Bratislava, Alfa 1990, ISBN 0-1554-9173.
    MICHALČÁK, O. – ADLER, E.: Výskum stability dunajských hrádzí. In: *Zborník vedeckých prác Stavebnej fakulty SVŠT.* Bratislava: Edičné stredisko SVŠT 1976, s. 17-28. ISBN 0-3552-5214.
    ŠÜTTI, J.: Určovanie priestorových posunov stavebných objektov. *Geodetický kartografický obzor.* 2000, roč. 2, č. 3, s. 8-16. ISSN 0811-6900.
    Article 18. Technical Cooperation. http://www.lac.uk/iso/tc456 (2013-09-28)
11. Za jazykovú a terminologickú správnosť záverečnej práce zodpovedá diplomant.
12. Formu postera (elektronická alebo aj tlačená) určí garant študijného programu.
13. Vzor pre poster je uvedený na dokumentovom serveri v akademickom informačnom systéme univerzity.

.....
podpis garanta študijného programu

Ustanovenia týchto pokynov som vzal na vedomie. Som si vedomý(á), že ak nebude moja diplomová práca vypracovaná v súlade s týmito pokynmi, nebude prijatá na obhajobu.

V Bratislave ....
podpis študenta

Čestne prehlasujem, že som bakalársku prácu Grafové algoritmy na hľadanie najkratších ciest v grafoch a ich aplikácia v spracovaní obrazu vypracoval samostatne s použitím citovanej literatúry a s odbornou pomocou vedúceho práce.

.....

Richard Rožnovják

Ďakujem vedúcej práce, Mgr. Márii Ždímalovej, PhD., za cenné pripomienky, rady, ochotu a venovaný čas pri vypracovávaní predkladanej práce.

Abstrakt

Segmentácia obrazu je známym problémom v oblasti spracovania obrazu. V posledných desaťročiach vzniklo množstvo metód založených na rôznych prístupoch k tomuto problému. Jedným z týchto prístupov je využitie poznatkov z teórie grafov. V našej práci sa zameriavame na segmentáciu pomocou najkratších ciest v grafe. Konkrétne sa venujeme metóde "Intelligent scissors", ktorá využíva na hľadanie najkratších ciest Dijkstrov algoritmus. Cieľom práce je aj implementovanie tejto metódy a jej následné praktické zhodnotenie.

Kľúčové slová: segmentácia obrazu, teória grafov, Dijkstrov algoritmus, najkratšia cesta v grafe, Intelligent scissors

Abstract

Segmentation techniques are very famous in image processing. There arises a lot of approaches based to this topic in the recent years. One of the approach is considering graphs and graphs algorithms. We focus in this final thesis on segmentation looking for and searching for the shortest paths in the graph. We consider Intelligent scissors with Dijkstra algorithm implementation. The aim of the final thesis is studying, implementation of Dijsktra's algorithm in Intelligent scissors and its application.

Key words: image segmentation, graph theory, Dijkstra's algorithm, shortest path of a graph, Intelligent scissors

1. Úvod

Segmentácia obrazu je už desaťročia predmetom skúmania mnohých odborníkov. No doposiaľ sa nepodarilo nájsť univerzálne riešenie tohto problému. Výskumníci k tomuto problému pristupujú rôznymi metódami a technikami, využívajúc poznatky z rôznych vedeckých odborov. Základné metódy a ich hlavné myšlienky popisujeme v druhej kapitole tejto práce. Jedným z týchto prístupov je aj využitie poznatkov teórie grafov. Prístup k segmentácii obrazu, kedy je obraz reprezentovaný pomocou grafu, si zhrnieme v kapitole tri. V práci sa venujeme segmentácii založenej na hľadaní najkratších ciest v grafe. Kapitola štyri je zameraná na tri najznámejšie algoritmy, pomocou ktorých tieto najkratšie cesty vieme nájsť. V práci sa zameriavame na segmentačnú metódu "Intelligent scissors". Je to interaktívna metóda, ktorá zapája do procesu segmentácie jej užívateľa. Na hľadanie najkratších ciest využíva Dijkstrov algoritmus. Čitateľ sa o nej viac dozvie v piatej kapitole tejto práce. Implementácii tejto metódy a popisu užívateľského prostredia programu sa venujeme v kapitole šesť. V poslednej kapitole testujeme túto metódu našou implementáciu a dosiahnuté výsledky zhŕňame v závere tejto práce.

2. Segmentácia obrazu

Segmentácia obrazu je klasickým problémom spracovania obrazu, ktorý sa skúma posledné desaťročia [1]. Jej hlavným cieľom je rozdelenie obrazu na disjunktné regióny tvorené pixelmi, pričom každý takýto región ohraničuje zmysluplný objekt alebo plochu nachádzajúcu sa v obraze. Takto vytvorené regióny zvykneme nazývať aj segmenty.

Jedným z hlavných problémov segmentácie obrazu je nejednoznačná obrazová informácia, často sprevádzaná šumom [2]. Čím viac apriórnej informácie je k dispozícii pre proces segmentácie, tým je možné dosiahnuť lepší výsledok. Segmentáciu možno rozdeliť na:

1. Úplnú segmentáciu – rozdelenie obrazu na neprekrývajúce regióny, ktoré zodpovedajú konkrétnym objektom alebo plochám obrazu.
2. Čiastočnú segmentáciu – rozdelenie obrazu na regióny, ktoré sú homogénne vzhľadom na určitú vlastnosť, ako napríklad jas, farbu, textúru a pod.

Vzniklo množstvo segmentačných algoritmov, ktoré pristupujú k tomuto problému rôznym spôsobom. Tieto metódy možno rozdeliť do piatich skupín [2].

1. Prahovanie
2. Hranovo orientované metódy
3. Metódy založené na regiónoch
4. Metódy založené na porovnávaní
5. Modely aktívnych kontúr

Prahovanie:

Je to najjednoduchšia segmentačná technika, ktorá je výpočtovo nenáročná a rýchla. Na segmentáciu objektov a pozadia sa využíva konštantná hodnota jasu nazvaná prah. Tieto metódy sa snažia určiť hodnotu prahu automaticky. Hodnota prahu môže byť konštantná pre celý obraz (globálne prahovanie) alebo sa môže meniť v rôznych častiach obrazu (lokálne prahovanie). Iba v špecifických prípadoch je postačujúce použitie globálneho prahovania. Okrem spomenutých dvoch základných typov prahovania existuje množstvo ďalších špecifikácií[2].

Hranovo orientované metódy:

Sú založené na hľadaní hrán pomocou hranových detektorov. Tieto hrany označujú miesta, v ktorých dochádza k zmene v úrovni šedej, vo farbe, v textúre a pod. Najčastejším problémom tohto prístupu, spôsobeným šumom alebo nevhodnou informáciou v obraze, je nachádzanie hrán v nevhodných miestach alebo absencia hrán v miestach, kde by sa mali nachádzať [2].

Metódy založené na regiónoch:

Tieto metódy nehľadajú hranice ohraničujúce regióny, ale samotné regióny tvorené množinou pixelov. Tieto metódy sú vhodné v prípade zašumených obrazov, kedy je náročné nájsť hranice medzi objektmi. Základná myšlienka týchto metód je rozdelenie obrazu na regióny s čo najväčšou homogenitou, pričom kritérium pre homogenitu môže byť založené na úrovni šedej, farbe, textúre a pod [2].

Metódy založené na porovnávaní:

Používajú sa na lokalizovanie objektov, ktoré sú podobné nami určenému vzoru. Zhodu medzi objektom a vzorom určuje kritérium optimality založené na vlastnostiach objektu. Hodnota kritéria optimality sa vypočíta pre každú polohu a otočenie vzoru v obraze. Ak táto hodnota presahuje určitý prah, daná poloha predstavuje umiestnenie vzoru [2].

Modely aktívnych kontúr:

V okolí objektu ktorý chceme segmentovať umiestnime počiatočnú krivku. Krivka sa následne deformuje tak, aby minimalizovala energetický funkcionál. Tento funkcionál je vytvorený ako suma vnútornej a vonkajšej energie. Vnútorná energia je určená tvarom krivky. Hladká krivka bez výrazných zmien tvaru nadobúda nízku hodnotu. Vonkajšia energia závisí od obrazu a nadobúda nízke hodnoty na hranách, kde gradient nadobúda najväčšie hodnoty [2].

Ďalším možným rozdelením segmentačných metód je rozdelenie na:

1. Automatické metódy
2. Interaktívne metódy

Automatické metódy nevyžadujú interakciu od užívateľa počas procesu segmentácie. Segmentácia týmito metódami je preto rýchlejšia a pohodlnejšia. Tieto metódy avšak nemusia vždy dávať požadované výsledky, ako to môže byť napríklad v prípade medicínskych dát, v ktorých sú objekty nejednoznačne definované, alebo sa v nich nachádza silný šum. Interaktívne metódy môžu tento problém prekonať zainteresovaním človeka do procesu segmentácie. Nevýhodou je dlhší segmentačný čas a potreba účasti osoby v procese segmentovania [1].

3. Segmentácia a teória grafov

Mnohé z metód segmentovania obrazu využívajú poznatky z teórie grafov. V tejto časti práce si tieto metódy popíšeme. Avšak ešte predtým potrebujeme mať zadefinované niektoré pojmy [1].

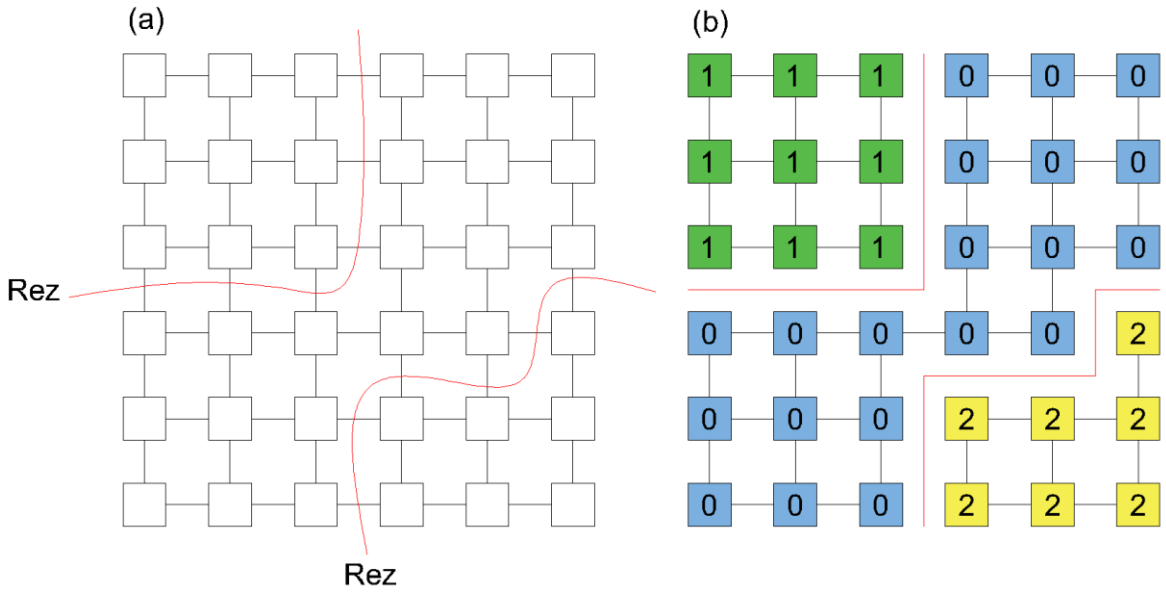
Nech $G=(V, E)$ je graf, pričom $V=\{v_1, v_2, \dots, v_n\}$ je množina vrcholov, ktoré môžu reprezentovať množinu pixelov obrazu alebo regióny v Euklidovskom priestore. E je množina hrán, ktoré spájajú susedné vrcholy. Každá hrana $(v_i,v_j) \in E$ je ohodnotená váhou $w(v_i,v_j)$, ktorá predstavuje určitú veličinu závislú napríklad od hodnôt v pixeloch, ktoré táto hrana spája.

Obraz možno rozdeliť na samostatné komponenty, pričom platí, že komponent A je súvislý graf $G'=(V',E')$, kde $V' \subseteq V$, $E' \subseteq E$ a E' obsahuje iba hrany z vrcholov V'. Teda neprázdne množiny $A_1,A_2, \dots, A_k$ tvoria rozdelenie grafu G ak $A_i \cap A_j = \emptyset$ pre $i \neq j$, i a $j \in (1, 2, \dots, k)$ a $A_1 \cup \dots \cup A_k = G$. Pre komponenty by malo platiť, že vlastnosti ako sú napríklad hodnota jasu, farba alebo textúra, sú podobné v rámci celého komponentu. Potom stupeň odlišnosti dvoch komponentov možno vypočítať ako rez grafu. Takýto rez prechádza hranami v grafe, ktoré rozdelia graf na dve disjunktné množiny A a B. Rez grafu je definovaný ako

$$cut(A, B) = \sum_{u \in A, v \in B} w(u, v) \qquad 3.1)$$

kde u a v sú vrcholy z odlišných komponentov.Potom ku problému segmentácie obrazu môžeme pristupovať ako k optimalizačnému problému, kedy sa snažíme minimalizovať určité kritérium. V tomto prípade bude optimálne rozdelenie grafu na dva segmenty to, ktoré minimalizuje rez grafu.

Segmentácia obrazu býva často formulovaná aj ako problém, v ktorom priraďujeme hodnoty z množiny L k prvkom z množiny lokalít S. Príkladom môže byť množina $L=\{objekt, pozadie\}$ a množina S tvorená množinou pixelov obrazu. Takto bude obraz rozdelený na dva segmenty a každý pixel bude jednoznačne priradený do jedného z dvoch segmentov [1].



Obr. 3.1

(a) Znázornenie dvoch rezov v grafe. Rezy rozdelia obraz na tri segmenty

(b) Priradenie hodnôt z množiny L={0, 1, 2} všetkým vrcholom (pixelom). Obraz je opäť rozdelený na tri segmenty.

V nasledujúcom texte popíšeme základné metódy segmentovania, ktoré využívajú poznatky z teórie grafov [1].

Metódy založené na minimálnej kostre grafu

Kostra T grafu G je strom, pričom platí $T=(V, E')$, kde $E' \subseteq E$. Graf môže mať viacero rozdielnych kostier, avšak minimálna kostra je kostra, ktorá má najmenšie váhy na hranách spomedzi všetkých kostier. Nájsť takúto minimálnu kostru grafu môžeme napríklad Primovým algoritmom, v ktorom iteratívne pridávame do stromu hrany s najmenšími váhami. Hlavná myšlienka týchto metód spočíva vo vlastnosti minimálnej kostry. Tá je taká, že zoskupuje pixely s podobnými hodnotami do podstromov. Tieto podstromy získame ak minimálnu kostru budeme rozpojovať v miestach, kde hrany nadobúdajú najvyššie váhy [1].

Metódy založené na minimálnych rezoch

Okrem rezu grafu definovaného v rovnici (3.1) existujú aj alternatívne definície rezov, ktorých hodnotu chceme minimalizovať. Z predošlého vyplýva výhoda týchto metód, a to možnosť použiť pre rôzne aplikácie rôzne druhy rezov. Hlavnou nevýhodou rezu definovaného ako (3.1) je, že uprednostňuje nachádzanie malých komponentov. Tento problém môžeme riešiť použitím normalizovaného rezu

$$Ncut(A,B) = \frac{cut(A,B)}{vol(A)} + \frac{cut(A,B)}{vol(B)} \qquad 3.2)$$

kde výraz $vol(A) = \sum_{v_i \in A, v_j \in V} w(v_i, v_j)$ predstavuje sumu váh medzi všetkými vrcholmi množiny *A* a všetkými vrcholmi grafu. Ak sa medzi vrcholmi nenachádza hrana, použije sa váha rovná nule. Pre takto definovaný rez platí, že ak množina *A* alebo *B* obsahuje málo vrcholov, hodnota rezu nebude nízka. Segmentovanie takto definovaným rezom má taktiež svoju nevýhodu, ktorou je vytváranie segmentov s podobnou sumou váh na hranách. Ďalšia definícia rezu využíva informáciu o hranici a vnútre segmentu. Konkrétne tento rez je definovaný ako

$$RegionCut(A,B) = \frac{cost(P)}{weight(P)} \qquad 3.3)$$

kde *P* je orientovaná cesta (viac o ceste v grafe sa čitateľ dočíta v [3] alebo v kapitole 4 tejto práce) v grafe G začínajúca a končiaca v tom istom vrchole, *cost(P)* je dĺžka cesty *P*, a *weight(P)* je suma váh v oblasti ohraničenej cestou *P*. Hlavnou nevýhodou tejto definície rezu je uprednostňovanie veľkých objektov a nutnosť segmentovať iba objekty s uzavretou hranicou. Okrem spomenutých rezov existujú aj ďalšie definície [1].

Grafové rezy založené na Markovových náhodných poliach

Štúdium psychológie ukázalo, že na interpretovanie obrazu je potrebné brať do úvahy aj kontext obrazovej informácie. To znamená, že ak sa budeme snažiť identifikovať objekt, ktorý celý nevidíme, nemusí sa nám to podariť. Obraz by sa mal preto chápať v priestorovom aj vo vizuálnom kontexte. Takýto prístup umožňuje teória Markovových náhodných polí [1].

Metódy založené na najkratších cestách v grafoch

Hľadanie najkratšej cesty (o najkratších cestách v grafe sa čitateľ dočíta viac v [3] alebo v kapitole 4 tejto práce) medzi dvoma vrcholmi je klasický problém teórie grafov. Najznámejším algoritmom, ktorý rieši tento problém, je Dijkstrov algoritmus. Ten spolu s ďalšími si bližšie popíšeme v kapitole 4. Problém segmentácie, v ktorom hľadáme najlepšiu hranicu segmentu, je preformulovaný ako problém hľadanie najkratšej cesty medzi dvoma vrcholmi. V praktických aplikáciách sa využíva interakcia od užívateľa, čím sa zvýši presnosť segmentácie. Používa sa "live-wire" metóda, ktorá umožní užívateľovi zvoliť prvý bod na hranici. Následne sa v reálnom čase zobrazí najkratšia cesta medzi prvým bodom a bodom, kde sa práve nachádza kurzor myši. Úlohou užívateľa je zvoliť polohu kurzoru myši tak, aby vykreslená najkratšia cesta čo najlepšie aproximovala časť hranice. Z tohto bodu sa následne budú vykresľovať nové najkratšie cesty a tento postup iteruje pokým sa nevytvorí uzavretá hranica. Vytvorená hranica je reprezentovaná postupnosťou orientovaných hrán medzi pixelmi, pričom každá hrana je ohodnotená hodnotou. Táto hodnota je nízka, ak je hrana vhodná ako hranica objektu. Aby cesta medzi dvoma bodmi bola najkratšia, musí prechádzať práve hranami s najmenšími hodnotami. Tým pádom vykresľované najkratšie cesty budú prechádzať hranicou objektu. Tento prístup k segmentácii vyžaduje v každom kroku hľadanie najkratších ciest z daného bodu do všetkých ostatných bodov. To je zároveň hlavnou nevýhodou, keďže je potrebný určitý výpočtový čas, ktorý narastá s počtom pixelov obrazu. V kapitole 5 si bližšie popíšeme segmentačnú techniku nazvanú "Intelligent scissors", založenú na princípe hľadania najkratších ciest v grafe.

4. Algoritmy hľadajúce najkratšie cesty v grafoch

Na úvod tejto kapitoly zadefinujeme pojmy z teórie grafov, ktoré využijeme pri popise algoritmov.

Definícia: Majme graf $G(V, E)$. Sledom grafu G nazývame postupnosť vrcholov $v_1, v_2, \dots, v_k$, ak platí $v_1, v_2, \dots, v_k \in V$ a hrana $(v_i, v_{i+1}) \in E$ pre $1 \leq i < k$, [3]. Tento sled sa nazýva cestou, keď $v_i \neq v_j$ pre každé i a j spĺňajúce $1 \leq i < j < k$. Navyše, ak pre sled platí ,že $k \geq 3$ a $v_i = v_j$ práve vtedy, keď $i = 0$ a $j = k$ pre $1 \leq i < j < k$, hovoríme o kružnici. Ak takáto kružnica má zápornú sumu váh na hranách, hovoríme o zápornej kružnici.

Problém najkratšej cesty je problémom nájdenia najkratšej cesty z určitého počiatočného vrcholu do iného koncového vrcholu v grafe [4]. V porovnaní so všetkými cestami medzi dvoma danými vrcholmi pre najkratšiu cestu platí, že má najnižšiu sumu cez všetky váhy na hranách, ktorými prechádza . Na graf je kladená podmienka, aby jeho hrany boli orientované a ohodnotené váhou. V reálnej aplikácii môžu vrcholu grafu reprezentovať mestá, orientované hrany grafu jednosmerné cesty a neorientované hrany cesty prechodné oboma smermi. V takomto prípade váhy na hranách môžu predstavovať čas, ktorý je potrebný pre cestu medzi mestami. Vyriešením problému najkratšej cesty medzi dvoma konkrétnymi vrcholmi získame cestu, ktorá zaberie najmenej času pri preprave medzi dvoma mestami, ktorým prislúchajú dané dva vrcholy, a to zo všetkých možných ciest. Najkratších ciest môže existovať viacero. Pre všetky však bude platiť že suma cez všetky váhy na hranách, ktoré cesta obsahuje, bude rovnaká. Líšiť sa môžu iba v počte hrán a vrcholmi, ktorými prechádzajú. Existuje mnoho algoritmov riešiacich tento problém. V tejto kapitole si uvedieme tri najznámejšie a najčastejšie používané. Ešte predtým však uvedieme ďalšie označenia a pseudokód funkcií, ktoré budeme pri popise algoritmov využívať [5].

Definujme váhu $w(p)$ cesty $p=\{v_0, v_1, \dots, v_k\}$ ako sumu váh na hranách, ktorými cesta prechádza.

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i) \tag{4.1}$$

Ďalej nech $\delta(u, v)$ je váha najkratšej cesty medzi vrcholmi u a v. Pre každý vrchol grafu v definujme hodnotu $d(v)$ ako hornú hranicu váhy najkratšej cesty z vrcholu s do vrcholu v. Horná hranica najkratšej cesty sa označuje aj ako odhad najkratšej cesty. Navyše nech $\pi(v)$ je ukazovateľ alebo identifikátor vrcholu, ktorý sa v najkratšej ceste nachádza pred vrcholom v. Okrem dĺžky najkratšej cesty nás často zaujíma aj samotná cesta, teda postupnosť vrcholov alebo hrán. Práve na to slúži identifikátor predošlého vrcholu π, pomocou ktorého sa vieme spätne dostať z ktoréhokoľvek vrcholu do vrcholu s a tým vytvoriť hľadanú najkratšiu cestu. Na začiatku dvoch algoritmov na hľadanie najkratších ciest treba nastaviť niektoré hodnoty na počiatočné. Na toto nech slúži funkcia *INITIALIZE*.

Peudokód funkcie INITIALIZE:

```
INITIALIZE [G(V, E), s]
1   for each vertex v∈V
2       d(v)=∞
3       π(v)=NIL
4   d(s)=0
```

Pod pojmom relaxácia hrany (u, v) budeme rozumieť testovanie, či dokážeme znížiť predbežne vypočítaný odhad najkratšej cesty, ak by prechádzala práve vrcholom u. Pseudokód funkcie vykonávajúcej túto činnosť vyzerá nasledovne:

```
RELAX[u, v, w]
1   if d(v)>d(u)+w(u, v)
2       d(v)=d(u)+w(u, v)
3       π(v)=u
```

kde parameter w je funkcia váh $w:E\rightarrow\mathbb{R}$, ktorá každej hrane priradí váhu.

4.1 Dijkstrov algoritmus

Dijkstrov algoritmus rieši problém nájdenia najkratších ciest v hranovo ohodnotenom a orientovanom grafe *G=(V, E)*, pričom všetky jeho hrany musia mať nezáporné váhy [5]. Takto nájdené najkratšie cesty nájde z jedného konkrétneho vrcholu grafu do všetkých ostatných vrcholov. Algoritmus si udržuje množinu vrcholov *S*, ktorých dĺžka najkratšej cesty bola už stanovená. Algoritmus opakovane vyberá vrchol $u \in V\text{-}S$ s minimálnou dĺžkou najkratšej cesty, pridá vrchol *u* do množiny *S*, a zrelaxujú sa všetky hrany smerujúce z vrcholu *u*. V nasledovnej implementácii sa uvažuje prioritná fronta vrcholov *Q*, ktorej priorita je určená minimálnou hodnotou *d(.)*. To znamená, že vrcholy v nej budú zoradené vzostupne podľa hodnoty odhadu najkratšej cesty, pričom na začiatku fronty je vrchol s minimálnou hodnotou odhadu najkratšej hrany.
Pseudokód Dijkstrovho algoritmu vyzerá nasledovne:

```
Dijkstra[G(V, E), w, s]
1   INITIALIZE[G(V, E), s]
2   S=∅
3   Q=V
4   while Q≠∅
5     u=EXTRACT_MIN(Q)
6     S=S∪{u}
7     for each vertex v∈neighbors(u)
8       RELAX[u, v, w]
```

Na začiatku algoritmu prebehne inicializácia veličín *d(.)* a *π(.)*. Následne sa nastaví množina *S* na prázdnu množinu a prioritná fronta sa naplní vrcholmi *V*. Počas behu programu platí *Q=V-S*. Vo **while** cykle sa z prioritnej fronty vyberie vrchol *u* s najmenšou hodnotou *d(.)*, ktorý sa následne uloží do množiny *S*. Prvým vybraným vrcholom bude vždy zvolený počiatočný vrchol *s*, z ktorého chceme vypočítať najkratšie cesty do ostatných vrcholov grafu. V riadku 7 a 8 prebehne relaxácia všetkých hrán smerujúcich z vrcholu *u*. Teda ak prechod cez vrchol *u* do vrcholu *v* zníži hodnotu *d(v)*, aktualizujeme hodnoty *d(v)* a *π(v)*. Algoritmus nikdy nevkladá nové hodnoty do fronty *Q* za riadkom 3, teda každý vrchol je vybratý z fronty *Q* a vložený do množiny *S* iba raz. Z toho vyplýva, že cyklus **while** bude mať presne $|V|$ iterácií. Dĺžka výpočtového času algoritmu závisí od

toho, ako je implementovaná prioritná fronta. Ak je fronta implementovaná ako pole, výpočtová zložitosť algoritmu je $O(|V|^2)$. Pri použití Fibonacciho haldy ako prioritnej fronty je možné dosiahnuť výpočtovú zložitosť $O(|E|+|V|log|V|)$.

4.2 Bellman-Fordov algoritmus

Bellman-Fordov algoritmus podobne ako Dijkstrov algoritmus rieši problém hľadania najkratších ciest z určitého vrcholu *s* [5]. Avšak oproti Dijkstrovmu algoritmu rieši túto úlohu aj pre grafy, v ktorých váhy vrcholov môžu byť ohodnotené zápornou hodnotou, ale graf nesmie obsahovať zápornú kružnicu. Záporná kružnica by totiž neustále znižovala dĺžku najkratšej cesty. Algoritmus vracia boolovskú hodnotu podľa toho, či sa v grafe nachádza záporná kružnica dosiahnuteľná z počiatočného bodu *s*. Ak sa v grafe takáto kružnica nenachádza, algoritmus nájde najkratšie cesty z bodu *s*, vypočíta dĺžky najkratších ciest a ako návratovú hodnotu vráti *true*. V opačnom prípade vráti hodnotu *false*, čo znamená ,že riešenie neexistuje.
Pseudokód Bellman-Fordovho algoritmu:

```
Bellman-Ford[G(V, E), w, s]
1   INITIALIZE[G(V, E), s]
2   for i = 1 to |V| - 1
3      for each edge (u, v) ∈ E
4         RELAX(u, v, w)
5   for each edge (u, v) ∈ E
6      if d(v) > d(u) + w(u, v)
7         return FALSE
8   return TRUE
```

Algoritmus začína inicializáciou veličín $d(.)$ a $\pi(.)$. Následne sa $|V| - 1$ krát vykoná relaxácia všetkých hrán v grafe. V riadkoch 5 až 8 sa zistí, či graf obsahuje zápornú kružnicu a podľa toho vráti prislúchajúcu návratovú hodnotu. Výpočtová zložitosť Bellman-Fordovho algoritmu je $O(|V||E|)$.

4.3 Floyd-Warshallov algoritmus

Floyd-Warshallov algoritmus rieši problém nájdenia najkratších ciest medzi všetkými pármi vrcholov v grafe $G = (V, E)$. Podobne ako pre Bellman-Fordov algoritmus, graf G môže obsahovať hrany so zápornou váhou, avšak nesmie obsahovať zápornú kružnicu. Algoritmus využíva ako vstup maticu susednosti $W = (w_{ij})$, ktorá je typu $n \times n$ a je definovaná ako

$$w_{ij} = \begin{cases} 0 & ak\ i = j \\ w(i,j) & ak\ i \neq j\ \wedge (i,j) \in E \\ \infty & ak\ i \neq j\ \wedge (i,j) \notin E. \end{cases} \qquad (4.3.1)$$

Algoritmus pracuje s vnútornými vrcholmi najkratšej cesty, pričom *vnútorný vrchol* cesty $p=\{v_1, v_2, \dots, v_l\}$ je ktorýkoľvek vrchol cesty okrem vrcholov v_1 a v_l, teda vrchol z množiny $\{v_2, v_3, \dots, v_{l-1}\}$. Označme vrcholy grafu celými číslami od 1 po n. Teda vrcholy grafu G tvoria množinu $\{1, 2, \dots, n\}$. Majme podmnožinu vrcholov $\{1, 2, \dots, k\}$ pre nejaké k. Pre každý pár vrcholov i a $j \in V$ uvažujeme všetky cesty z vrcholu i do j také, ktorých vnútorné vrcholy sú z množiny $\{1, 2, \dots, k\}$, a nech p je cesta s minimálnou sumou váh na hranách zo všetkých takýchto ciest. Floyd-Warshallov algoritmus využíva vzťah medzi cestou p a najkratšími cestami z vrcholu i do j ktorých vnútorné vrcholy sú z množiny $\{1, 2, \dots, k - 1\}$. Ak vrchol k nie je vnútorným vrcholom cesty p, potom všetky vnútorné vrcholy cesty p sú z množiny $\{1, 2, \dots, k - 1\}$. Teda, najkratšia cesta z vrcholu i do j s vnútornými vrcholmi z množiny $\{1, 2, \dots, k - 1\}$ je zároveň najkratšou cestou z vrcholu i do j s vnútornými vrcholmi z množiny $\{1, 2, \dots, k\}$. Ak k je vnútorným vrcholom cesty p, p môžeme rozložiť na cestu p_1 z vrcholu i do k a druhú cestu p_2 z k do j. Pre najkratšie cesty v grafe platí tvrdenie, že ak $p = \{v_1, v_2, \dots, v_k\}$ je najkratšia cesta z vrcholu v_0 do v_k, potom aj cesta $p_{ij} = \{v_i, v_{i+1}, \dots, v_j\}$ pre $0 \leq i < j \leq k$ je najkratšou cestou z vrcholu v_i do v_j. Na základe tohto tvrdenia je cesta p_1 najkratšou cestou z vrcholu i do k so všetkými vnútornými vrcholmi z množiny $\{1, 2, \dots, k\}$. V skutočnosti môžeme učiniť ešte silnejšie tvrdenie. Pretože vrchol k nie je vnútorným vrcholom cesty p_1, všetky vnútorné vrcholy cesty p_1 sú z množiny $\{1, 2, \dots, k - 1\}$. Teda, cesta p_1 je najkratšou cestou z vrcholu i do k so všetkými vnútornými vrcholmi z množiny $\{1, 2, \dots, k - 1\}$. Podobne, cesta p_2 je najkratšou cestou z vrcholu k do j so všetkými vnútornými vrcholmi z množiny $\{1, 2, \dots, k - 1\}$. Nech $d_{ij}^{(k)}$ je váha najkratšej cesty medzi vrcholmi i a j, ktorej všetky

vnútorné vrcholy sú z množiny *{1, 2, ... , k}*. Ak *k = 0*, cesta z vrcholu *i* do *j* bez vnútorných bodov s číslom väčším ako 0 nemá vôbec žiadne vnútorné vrcholy. Taká cesta má najviac jednu hranu, teda $d_{ij}^{(0)} = w_{ij}$. Z predošlého môžeme $d_{ij}^{(k)}$ definovať rekurzívne ako

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & ak\ k = 0 \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}), & ak\ k \geq 1. \end{cases} \qquad (4.3.2)$$

Pretože pre každú cestu sú vnútorné vrcholy z množiny *{1, 2, ... , n}*, matica $D^{(n)} = d(_{ij}^{(n)})$ je konečným riešením. Platí $d_{ij}^{(n)} = \delta(i, j)$ pre všetky $i, j \in V$. V algoritme využívame rekurzívny vzťah (4.3) a hodnoty $d_{ij}^{(k)}$ počítame zvyšovaním hodnoty *k*. Výstupom algoritmu je matica $D^{(n)}$ váh najkratších ciest.

Pseudokód Floydovho-Warshallovho algoritmu:

```
Floyd-Warshall[W]
1   n = num_of_rows(W)
2   D(0) = W
3   for k = 1 to n
4      let D(k) = (d_ij(k)) be a new n × n
5   for i = 1 to n
6      for j = 1 to n
7         D_ij(k) = min(D_ij(k-1), D_ik(k-1) + D_kj(k-1))
8   return D(n)
```

Výpočtová zložitosť Floydovho-Warshallovho algoritmu je $\Theta(n^3)$. Výsledná matica *D* nám poskytne informáciu o váhach najkratších ciest, avšak nie o samotných cestách. Tento problém môžeme vyriešiť vypočítaním matice predchodcov *Π* počas výpočtu matice $D^{(k)}$ v algoritme. Vypočítame postupnosť matíc $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(n)}$, kde $\Pi = \Pi^{(n)}$ a zadefinujeme $\pi_{ij}^{(k)}$ ako predchodcu vrcholu *j* v najkratšej ceste z vrcholu *i* so všetkými vnútornými vrcholmi z množiny *{1, 2, ... , k}*. Hodnotu $\pi_{ij}^{(k)}$ definujeme rekurzívne.

Ak *k = 0*, najkratšia cesta z vrcholu *i* do *j* nemá žiadne vnútorné vrcholy.

Teda,

$$\pi_{ij}^{(0)} = \begin{cases} NIL & ak\ i = j\ \vee\ w_{ij} = \infty \\ \text{i} & ak\ i \neq j\ \wedge\ w_{ij} < \infty. \end{cases} \tag{4.3.3}$$

Pre $k \geq 1$,

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & ak\ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & ak\ d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases} \tag{4.3.4}$$

5. Segmentačná technika Intelligent scissors

V tejto kapitole popíšeme interaktívnu segmentačnú techniku nazvanú "Intelligent scissors", [6]. Je založená na hľadaní optimálnej cesty v grafe, a jej hlavnou výhodou je použitie nástroju "live-wire". Tento nástroj umožňuje užívateľovi interaktívne zvoliť časť segmentovanej hranice tak, že mu okamžite zobrazuje najkratšie cesty zo zvoleného pixelu do aktuálnej polohy kurzoru myši.

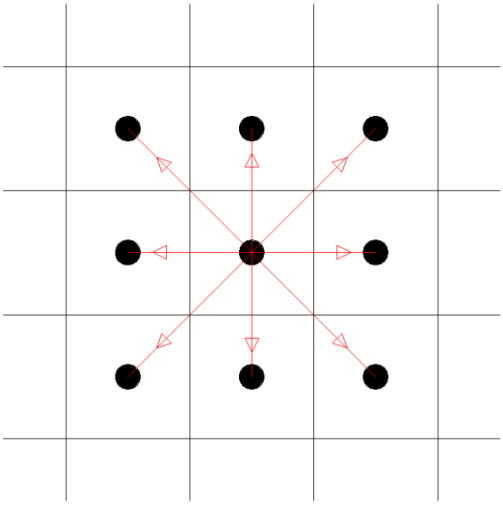
Najkratšie cesty sa nájdu pre všetky pixely v obraze, a užívateľ teda môže zvoliť tú najkratšiu cestu, ktorá najlepšie vystihuje hranicu segmentovaného objektu. Po tom, ako užívateľ zvolí najkratšiu cestu, sa z posledného pixelu tejto cesty opäť nájdu všetky najkratšie cesty a tento postup pokračuje, kým sa postupne nevytvorí uzavretý segment. Posledný zvolený pixel najkratšej cesty teda vždy tvorí nový počiatočný pixel pre novú časť segmentovanej hranice. Hranica zvolená v predošlých krokoch zostáva nemenná.

Aby sa minimalizovala potreba voliť nový pixel, ktorý tvorí koniec segmentovaného úseku, tento pixel sa generuje automaticky technikou "boundary cooling". Tá využíva toho, že väčšina najkratších ciest bude mať niekoľko začiatočných pixelov spoločných. Ako nová časť segmentovanej hranice sa teda zvolí množina pixelov, ktorú získame ako prienik určitého množstva najkratších ciest.

Voľba nového pixelu zahŕňa aj ďalší problém. Tým je nepresnosť, ktorá vzniká pri manipulácii s myšou a malými rozmermi pixelu v obraze. Tento problém sa rieši takzvaným prichytávaním kurzoru, kedy je ako nový pixel zvolený pixel z určitého okolia, v ktorom sa nachádza najväčšia hodnota veľkosti gradientu. Navyše technika "Intelligent

scissors" zahŕňa aj dynamický tréning. Ten umožňuje algoritmu prispôsobiť sa rôznym typom hrán.

Obraz je reprezentovaný grafom, kde každý pixel obrazu predstavuje vrchol grafu. Všetky hrany grafu sú ohodnotené váhou a orientované, pričom z každého vrcholu smeruje práve osem hrán k susedným vrcholom. Výnimkou sú iba vrcholy na hranách obrazu. Obrázok 5.1 ilustruje vrcholy a hrany v takomto grafe. Ukážka fungovania nástroja "live-wire" pri segmentovaní je na obrázku 5.2.



Obr. 5.1

Znázornenie pixelov a orientovaných hrán v grafe reprezentujúcom obraz

Obr. 5.2

(a) Určenie počiatočného bodu segmentácie a vykreslenie najkratšej cesty medzi týmto bodom a kurzorom myši.

(b) Po zmene polohy kurzoru myši sa zmenila aj vykreslená najkratšia cesta

(c) Zvolenie nového bodu a vykreslenie najkratšej cesty z tohto bodu k polohe kurzora myši. Vysegmentovaný úsek medzi prvým a druhým bodom už zostáva nemenný.

(d) Vykreslenie inej najkratšej cesty z druhého bodu.

5.1 Výpočet hodnôt váh na hranách

Ak p a q sú dva susedné pixely v obraze, potom *l(p, q)* je hodnota váhy na orientovanej hrane z pixelu p do q [6]. Funkcia *l(p, q)* je váženou sumou funkcií obrazových charakteristík, ktorými sú:

f_Z: Nulové body Laplaciánu

f_G: veľkosť gradientu

f_D: smer gradientu

f_P: hodnota pixelu

f_I: "vnútorná" hodnota pixelu

f_O: "vonkajšia" hodnota pixelu.

Výsledná funkcia *l(p, q)* má teda tvar:

$$l(p,q) = \omega_Z * f_Z(q) + \omega_G * f_G(q) + \omega_D * f_D(p,q) + \omega_P * f_P(q) + \qquad (5.1.1)$$

$$+\omega_I * f_I(q) + \omega_O * f_O(q)$$

kde každá ω je váha prislúchajúcej funkcie. Všetky tieto funkcie majú obor hodnôt interval $\langle 0,1 \rangle$. Empiricky bolo určené, že váhy $\omega_Z = 0.3$, $\omega_G = 0.3$, $\omega_D = 0.1$, $\omega_P = 0.1$, $\omega_I = 0.1$ a $\omega_O = 0.1$ dávajú dobré výsledky pre väčšinu obrazov. V prípade potreby môžu byť tieto váhy upravené, avšak ich suma musí byť rovná 1. Funkcie f_Z, f_G *a* f_D sa nazývajú statické. To znamená, že sú nemenné a nevyužívajú žiadnu apriórnu informáciu o obraze, teda ich funkčný predpis je nemenný. Naopak, funkcie f_G, f_P, f_I *a* f_O označujeme ako dynamické. Ich funkčný predpis je definovaný iba ak sa využíva tréningová vzorka dát, a od nej závisí. Teda ak sa nevyžíva tréning, sú ekvivalentné konštantnej nulovej funkcii. Funkcia f_G ako jediná je definovaná zároveň ako statická aj dynamická. Teda v prípade tréningu jej funkčný predpis závisí od tréningovej vzorky dát, v inom prípade je táto funkcia statická. Hlavnou myšlienkou pri konštruovaní týchto funkcií je to, aby ich návratová hodnota bola nízka, ak hranica segmentu prechádza práve medzi vrcholmi p a q. Tieto hrany tým pádom budú tvoriť najkratšie cesty vypočítané Dijkstrovým algoritmom.

Obrázok 5.1.1 ilustruje výsledné hodnoty váh na hranách grafu. Z každého pixelu vedie 8 hrán do susedných pixelov, preto je nemožné zobraziť hodnoty všetkých váh v jednom dvojrozmernom obrázku. Obrázok preto ilustruje iba hodnoty váh na hranách, ktoré smerujú iba k svojmu ľavému hornému susedovi. Čierna farba predstavuje najmenšiu nulovú váhu na hrane.

Obr. 5.1.1

Vľavo pôvodný obrázok. Vpravo ilustrácia váh na hranách grafu.

5.2 Nulové body Laplaciánu (f_Z)

Hlavným významom funkcie f_Z je detekcia hrán. Vykonáme konvolúciu obrazu s Laplaciánom Gaussiánu, kde šírka konvolučného jadra je určená hodnotou σ. Šírka jadra má vplyv na predvyhľadenie obrazu za účelom odstránenia šumu. Funkcia $f_Z(q)$ má tvar:

$$f_Z(q) = \begin{cases} 0 \; ak \; I_L(q) = 0 \\ 1 \; ak \; I_L(q) \neq 0 \end{cases} \tag{5.2.1}$$

kde $I_L(q)$ je Laplacián obrazu I v bode q. Ak sa pixel nachádza v nulovom bode (bod, v ktorom sa mení znamienko funkcie), funkcia f_Z nadobúda nulovú hodnotu v tomto pixeli. Teda ak sa pixel nachádza na hrane v obraze, prírastok k váhe hrany je nulový. Tým pádom najkratšie cesty v obraze budú prechádzať hranami v obraze z dôvodu nízkych hodnôt na hranách. Môžeme vykonať viacero kovolúcií s rôznymi šírkami konvolučných masiek. V takom prípade výsledná hodnota funkcie f_Z bude vážená suma

$$f_Z(q) = w_1 * f_Z^1(q) + w_2 * f_Z^2(q) + \cdots + w_N * f_Z^N(q) \tag{5.2.2}$$

kde N je počet rôznych konvolučných masiek, a suma váh je rovná 1. Funkcia f_Z teda nadobúda hodnotu z intervalu $\langle 0,1 \rangle$.

Obrázok 5.2.1 ilustruje aplikáciu funkcie f_Z na obrázok. Boli použité tri rôzne konvolučné masky. Čierna farba pixelov reprezentuje nulovú hodnotu funkcie f_Z. Biela farba naopak maximálnu.

Obr. 5.2.1
Vľavo pôvodný obrázok. Vpravo ilustrácia funkcie f_Z.

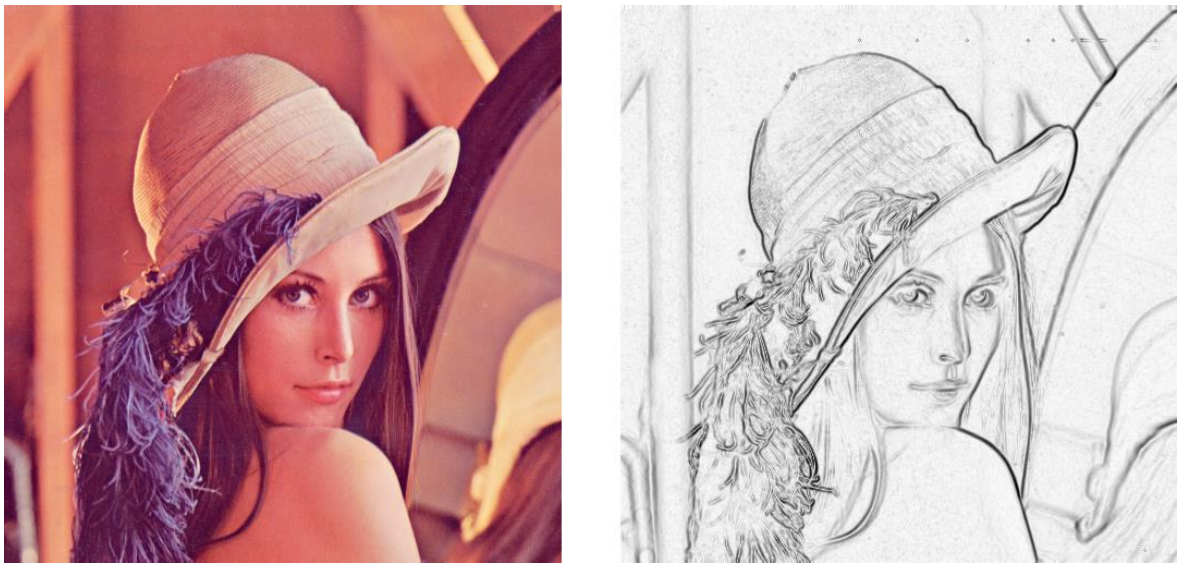
5.3 Veľkosť gradientu (f_G)

Hodnota funkcie f_Z nás informuje o tom, či daným pixelom prechádza hrana v obraze. Avšak nevieme, aká výrazná hrana to je. Na to slúži funkcia veľkosti gradientu $f_G(q)$. V tejto kapitole si popíšeme funkciu veľkosti gradientu ako statickú funkciu. Dynamickej funkcii sa budeme venovať v kapitole 5.7 venovanej tréningu tejto funkcie. Veľkosť gradientu vypočítame aproximovaním parciálnych derivácií obrazu v smere *x* a *y* pomocou derivácie Gaussovho jadra. Ak (I_x, I_y) je gradient obrazu, veľkosť gradientu vypočítame ako $G = \sqrt{I_x^2 + I_y^2}$. Na výrazných hranách veľkosť gradientu nadobúda veľké hodnoty. My avšak požadujeme, aby funkcia f_G nadobúdala na hranách práve nízke hodnoty kvôli tomu, aby najkratšie cesty v grafe prechádzali práve hranami obrazu. Preto funkcia $f_G(q)$ má tvar

$$f_G(q) = \frac{max(G(q)') - G(q)'}{\max(G(q)')} = 1 - \frac{G(q)'}{\max(G(q)')} \tag{5.3.1}$$

kde $G'=G-min(G)$. Funkcia $f_G(q)$ je lineárna klesajúca funkcia, ktorá nadobúda hodnoty z intervalu $\langle 0,1 \rangle$. Podobne ako pre f_Z, môžeme použiť viacero konvolučných masiek.

Avšak výslednú hodnotu veľkosti gradientu zvolíme ako maximum veľkostí gradientu, ktoré získame použitím viacerých rôznych konvolučných masiek. Obrázok 5.3.1 ilustruje aplikáciu funkcie f_G na obrázok. Opäť boli použité tri rôzne konvolučné masky na výpočet jednej výslednej funkcie. Čierna farba pixelov reprezentuje minimálne nulové váhy na hranách grafu.



Obr. 5.3.1
Vľavo pôvodný obrázok. Vpravo ilustrácia funkcie f_G.

5.4 Smer gradientu (f_D)

Pri vytváraní hranice segmentu chceme, aby hranica bola plynulá a bez výrazných zmien v smere. Toto zabezpečuje funkcia f_D tým, že jej prírastok k váhe hrany bude nízky, ak nedôjde k výraznej zmene v smere hrany tvoriacej segment. Označme *D(p)* jednotkový vektor v smere gradientu vo vrchole *p*. Ďalej nech *D'(p)* je jednotkový vektor kolmý na vektor *D(p)* (otočený o 90° v smere hodinových ručičiek). Teda ak $D(p) = (I_x, I_y)$, *potom* $D'(p) = (I_y, -I_x)$. Funkcia f_D má predpis

$$f_D(p,q) = \frac{2}{3\pi}\{acos[d_p(p,q)] + acos[d_q(p,q)]\} \tag{5.4.1}$$

kde

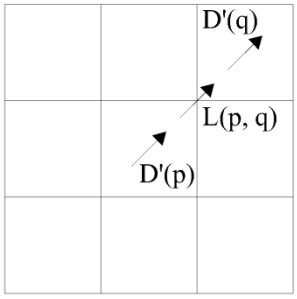
$$\begin{aligned} d_p(p,q) &= D'(p) \cdot L(p,q) \\ d_q(p,q) &= L(p,q) \cdot D'(q) \end{aligned} \tag{5.4.2}$$

a

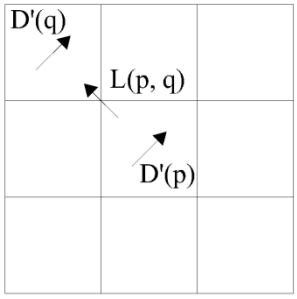
$$L(p,q) = \frac{1}{\|p-q\|}\begin{cases} q-p & ak\ D'(p)\cdot(q-p) \geq 0 \\ p-q & ak\ D'(p)\cdot(q-p) < 0 \end{cases} \qquad (5.4.3)$$

je jednotkový vektor, ktorý má smer z bodu p do q alebo naopak. Zvolí sa tá orientácia vektoru *L(p, q)*, ktorá minimalizuje uhol medzi vektorom *L(p, q)* a vektorom *D'(p)*. Funkcia f_D nadobúda nízku hodnotu, ak smer gradientu v dvoch susedných pixeloch je rovnaký, a zároveň rovnobežný k hrane medzi nimi.

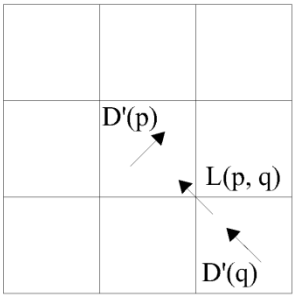
Obrázok 5.4.1 ilustruje tri rôzne prípady výpočtu hodnoty funkcie $f_D(p, q)$.



D'(p) = (0.707, 0.707)
D'(q) = (0.707, 0.707)
L(p, q) = (0.707, 0.707)
$d_p(p, q) = 1$
$d_q(p, q) = 1$
$f_D = 0$



D'(p) = (0.707, 0.707)
D'(q) = (0.707, 0.707)
L(p, q) = (-0.707, 0.707)
$d_p(p, q) = 0$
$d_q(p, q) = 0$
$f_D = 0.666$

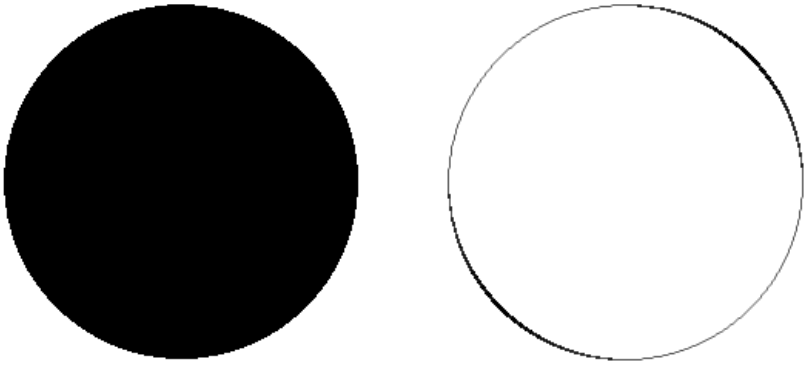


D'(p) = (0.707, 0.707)
D'(q) = (-0.707, 0.707)
L(p, q) = (0.707, -0.707)
$d_p(p, q) = 0$
$d_q(p, q) = -1$
$f_D = 1$

Obr. 5.4.1

Výpočet hodnoty funkcie f_D pre tri rôzne situácie.

Na obrázku 5.4.2 vľavo sa nachádza originály obraz. Na pravej strane sa nachádza obraz ktorý ilustruje vypočítané hodnoty funkcie $f_D(p, q)$ pre ľavých horných susedov. Čierna farba opäť reprezentuje nízke hodnoty váh na hranách. Vidno, že nízke hodnoty sú práve v miestach, kde smer gradientu dvoch susedných vrcholov (v tomto prípade pixelu a jeho ľavého horného susedného pixelu) je viac-menej rovnobežný s ich hranou ktorá ich spája.



Obr. 5.4.2

Vľavo pôvodný obrázok. Vpravo ilustrácia funkcie f_D pre hrany smerujúce k ľavému hornému susednému pixelu.

5.5 Hodnota pixelu (f_P), "vnútorná" a "vonkajšia" hodnota pixelu (f_I) a (f_O)

Tieto funkcie majú zmysel iba v prípade, keď sa využíva tréning. Ako tréningovú množinu musíme mať k dispozícii časť vysegmentovanej hrany tvorenú množinou pixelov. Funkcia f_P je jednoducho preškálovaná hodnota pixelu v obraze. Keďže obraz zvyčajne nadobúda hodnoty intenzity od 0 po 255, f_P je definovaná ako

$$f_P(p) = \frac{1}{255} I(p) \tag{5.5.1}$$

kde *I(p)* je hodnota intenzity pixelu p v obraze. Funkcie f_I a f_O sú definované podobne, avšak ich hodnota nie je intenzita v pixeli p, ale v pixeli, ktorý sa nachádza k pixelov od bodu p v smere gradientu, alebo v opačnom smere. Formálne

$$f_I(p) = \frac{1}{255} I(p + k * D(p)) \tag{5.5.2}$$

a

$$f_I(p) = \frac{1}{255} I\big(p - k * D(p)\big). \tag{5.5.3}$$

5.6 Farebný obraz

V tejto kapitole popíšeme, ako vypočítať váhy na hranách v prípade, ak je náš obraz farebný, teda každý pixel nesie informáciu o intenzite troch farebných kanálov. Funkcia f_Z a funkcia veľkosti gradientu sú vypočítané nezávisle pre tri farebné kanály. Výslednou hodnotou bude maximálna hodnota spomedzi týchto troch farebných kanálov. Pri výpočte ostatných funkcií sa RGB farba prevedie do farebného priestoru HSB (hue, saturation, brightness) a na výpočet využijeme hodnotu jasu (brightness).

5.7 Tréning

Pri segmentovaní najkratšie cesty vedú práve výraznými hranami v obraze. Môže však nastať situácia, kedy chceme segmentovať časť obrazu, ktorá je definovaná menej výraznými hranami. Ak sa v okolí takejto slabšie definovanej oblasti nachádza výrazná hrana, najkratšie cesty budú prechádzať touto hranou, čo spôsobí problémy pri segmentovaní. Takúto situáciu vidno na obrázku 5.8.1 v ďalšej kapitole. Za týmto účelom je možné použiť tréning funkcií f_G, f_P, f_I a f_O. Ako tréningová vzorka sa použijú pixeli z naposledy vysegmentovanej časti hranice. To umožní funkciám adaptovať sa na vlastnosti konkrétnej hrany. Prírastok týchto funkcií k výslednej váhe hrán v grafe bude mať za následok, že najkratšie cesty budú viesť práve hranami, ktoré sú podobné tréningovej vzorke. Pre efektívnosť výpočtu sa na začiatku segmentácie predpočítajú štyri pomocné obrázky I_G, I_P, I_I a I_O. Vypočítajú sa jednoducho preškálovaním a celočíselným zaokrúhlením hodnôt funkcií f'_G, f_P, f_I a f_O, kde funkcia $f'_G = G'/max(G')$ je preškálovaná hodnota veľkosti gradientu. Nech $n_P = n_I = n_O = 256$ a $n_G = 1024$, potom

$$\begin{aligned} I_P &= round((n_P - 1)f_P) \\ I_I &= round((n_I - 1)f_I) \\ I_O &= round((n_O - 1)f_O) \\ I_G &= round((n_G - 1)f'_G). \end{aligned} \tag{5.7.1}$$

Nech p je tréningová vzorka. Zvolíme jej maximálnu veľkosť a označíme ju ako *t*. Jej veľkosť je zvyčajne 32 až 64 pixelov. Je tvorená poslednými pixelmi v naposledy vysegmentovanej časti hranice. Posledné pixeli tejto množiny by mali mať väčšiu váhu, preto uvažujeme klesajúcu funkciu *w(i)*, kde *i* je poradie pixelov v tréningovej vzorke, pričom ako prvý pixel v tejto množine uvažujeme posledný pixel z naposledy vysegmentovanej časti hranice.

Keďže je tréning založený na vzorke dát a ovplyvňuje hodnotu váh na hranách v grafe, môže byť nežiadúci v prípade, kedy sa vlastnosti segmentovanej hranice menia. V takom prípade nemusí byť tréning zahrnutý pri výpočte váh na hranách grafu.

Ďalej vytvoríme štyri histogramy h_G, h_P, h_I a h_O. Nech $t_s \leq t$ je počet pixelov, ktoré máme k dispozícii pre tréning. Potom

$$\begin{aligned} h_G\big(I_G(p_i)\big) &= h_G\big(I_G(p_i)\big) + w(i) \\ h_P\big(I_P(p_i)\big) &= h_P\big(I_P(p_i)\big) + w(i) \\ h_I\big(I_I(p_i)\big) &= h_I\big(I_I(p_i)\big) + w(i) \\ h_O\big(I_O(p_i)\big) &= h_O\big(I_O(p_i)\big) + w(i) \end{aligned} \tag{5.7.2}$$

pre $i = 0, 1, \ldots, t_s - 1$.

Ako ďalšie vytvoríme funkcie m_G, m_P, m_I a m_O, ktorými nahradíme funkcie f_G, f_P, f_I a f_O. Nech M je maximálna celočíselná hodnota váhy, ktorú môže hrana nadobudnúť. Potom napríklad maximálny prírastok funkcie veľkosti gradientu k váhe bude $M_G = \omega_G \cdot M$. Funkcie m_G, m_P, m_I a m_O majú predpis

$$\begin{aligned} m_G &= round\left[\frac{\max(h_G) - h_G}{\max(h_G)} M_G\right] = round\left[M_G\left(1 - \frac{h_G}{\max(h_G)}\right)\right] \\ m_P &= round\left[M_P\left(1 - \frac{h_P}{\max(h_P)}\right)\right] \\ m_I &= round\left[M_I\left(1 - \frac{h_I}{\max(h_I)}\right)\right] \\ m_O &= round\left[M_O\left(1 - \frac{h_O}{\max(h_O)}\right)\right] \end{aligned} \tag{5.7.3}$$

kde delenie maximom histogramu preškáluje histogram, aby nadobúdal hodnoty z intervalu $\langle 0, 1 \rangle$, a odčítaním od hodnoty 1 tento histogram invertujeme.

Funkcia veľkosti gradientu je ako jediná definovaná pre prípad, kedy je tréning aktívny aj neaktívny. Preto vytvoríme novú funkciu m'_G ako kombináciu statickej a dynamickej funkcie. Táto kombinovaná funkcia veľkosti gradientu má predpis

$$m'_G(x) = \begin{cases} min\left\{m_G(x), round\left[M_G\left(1 - \frac{x(s - t_s)}{(n_G - 1)s}\right)\right]\right\} & ak\ t_s < s \\ m_G(x) & ak\ t_s \geq s \end{cases} \tag{5.7.4}$$

kde $x = 0, 1, \ldots, n_G - 1$, a s predstavuje minimálny počet pixelov v tréningovej vzorke, ktorý považujeme za dostatočný na vytvorenie dynamickej funkcie veľkosti gradientu.

Môže nastať niekoľko prípadov. V prípade, ak $t_s < s$, vytvorí sa kombinovaná funkcia veľkosti gradientu. Na veľkosť hodnoty s nie je kladená žiadna podmienka v závislosti od hodnoty t, tým pádom pre tento prípad môžeme uvažovať aj $t_s \leq t < s$. Pravá strana výrazu pre minimum je lineárna klusajúca funkcia, pre ktorú platí, že v bode 0 nadobúda hodnotu M_G, a v bode $n_G - 1$ hodnotu menšiu ako M_G a zároveň väčšiu ako 0, pričom táto hodnota závisí od hodnôt t_s a s. Ak $t_s \geq s$, výslednou funkciou bude funkcia m_G. Ak žiadne tréningové dáta nie sú k dispozícii alebo sa tréning neuvažuje, $t_s = 0$ a funkcia m'_G je rovná funkcii f_G definovanej v (5.3.1).

5.8 Finálna hodnota váh na hranách

Funkciou (5.1.1) sme popísali výpočet váh na hranách grafu. V nej funkcie f_G, f_D, f_Z, f_P, f_I a f_O nadobúdali hodnoty z intervalu $\langle 0, 1 \rangle$. Avšak ako finálnu budeme uvažovať upravenú funkciu

$$l'(p,q) = l_s(p,q) + w_N(p,q) \cdot m'_G\big(I_G(q)\big) + m_P\big(I_P(q)\big) + m_I\big(I_I(q)\big) + m_O(I_O(q)) \tag{5.8.1}$$

kde

$$l_s(p,q) = round[M_Z \cdot f_Z(q)] + round[M_D \cdot f_D(p,q)] \tag{5.8.2}$$

a

$$w_N(p,q) = \begin{cases} 1 & ak\ L_x(p,q) \neq 0 \wedge L_y(p,q) \neq 0 \\ \dfrac{1}{\sqrt{2}} & ak\ L_x(p,q) = 0 \vee L_y(p,q) = 0 \end{cases} \tag{5.8.3}$$

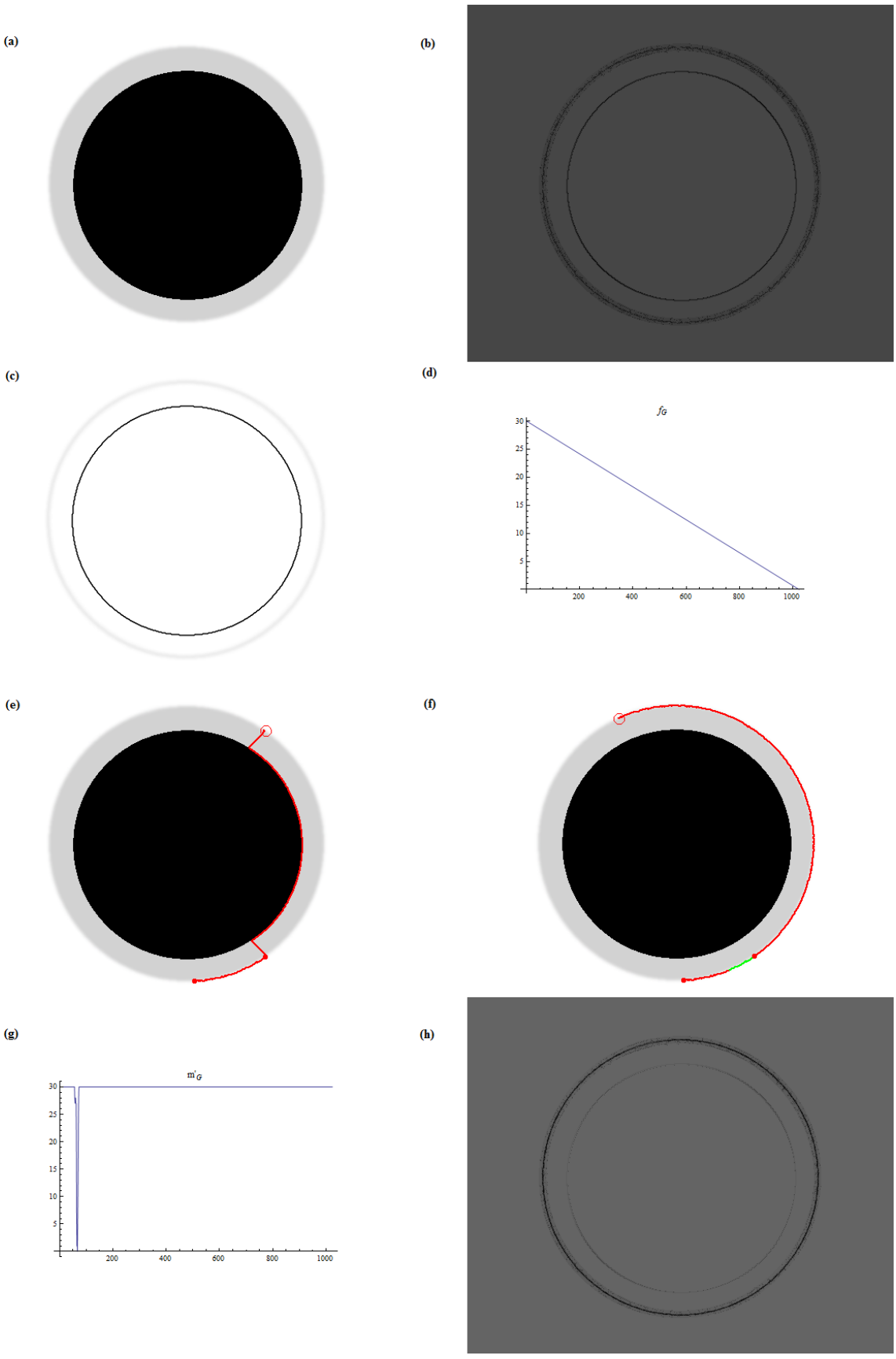
pričom funkcia $l_s(p, q)$ je funkciou statických váh na hranách, teda tých, ktoré sú nemenné bez ohľadu či je tréning aktívny alebo nie. Z dôvodu urýchlenia výpočtov sú tieto hodnoty vypočítané pre všetky váhy na hranách grafu ešte pred samotnou segmentáciou obrazu. Funkcia $w_N(p, q)$ zahŕňa do výpočtov výslednej hodnoty prírastku od funkcie veľkosti gradientu aj Euklidovskú vzdialenosť medzi vrcholmi. Hodnoty funkcií $L_x(p, q)$ a $L_y(p, q)$ pre dva susedné vrcholy p a q v grafe sú zložkami vektora, ktorý získame z vektorovej funkcie (5.4.3). Teda, ak pixel q je horizontálnym alebo vertikálnym susedom pixelu p, hodnota funkcie veľkosti gradientu sa nezmení. Ale v prípade, ak pixel q je diagonálnym

susedom pixelu *p*, hodnota funkcie veľkosti gradientu je prenásobená hodnotou $1/\sqrt{2}$. Ak tréning neuvažujeme, ako funkciu *l´(p, q)* zvolíme funkciu

$$l'(p,q) = l_s(p,q) + w_N(p,q) \cdot m'_G\big(I_G(q)\big) \tag{5.8.4}$$

a teda neuvažujeme funkcie f_P, f_I *a* f_O ktoré bez tréningu nemajú zmysel.
Funkcia *l´(p, q)* nadobúda celočíselné hodnoty z intervalu $\langle 0, M-1\rangle$. Výsledná hodnota váhy na hrane z pixelu *p* do *q* bude mať teda váhu *l´(p, q)*. Graf s vypočítanými váhami na hranách je vstupom do Dijkstrovho algoritmu, pomocou ktorého sa vypočítajú najkratšie cesty zo zvoleného pixelu do všetkých ostatných pixelov obrazu.

Obrázok 5.8.1 znázorňuje význam tréningu funkcií. V časti (a) je zobrazený pôvodný obraz. Naznačíme si priebeh segmentácie, ak chceme segmentovať vonkajšiu hranu šedého medzikružia. Táto hrana bola zámerne jemne rozmazaná. Na obrázku sú v časti (b) ilustračne znázornené výsledné váhy na hranách grafu. Časť (c) zobrazuje hodnoty funkcie veľkosti gradientu. Vďaka rozmazaniu vonkajšie hrany, táto hrana nadobúda menšie hodnoty veľkosti gradientu v porovnaní s hranou na rozhraní čierneho kruhu a medzikružia. Funkcia m'_G, ktorá je v prípade neaktívneho tréningu uvažovaná ako funkcia f_G definovaná ako (5.3.1), je zobrazená na obrázku v časti (d). Podľa definície veľká hodnota gradientu znamená malý prírastok k váhe na hrane. Tým pádom sú váhy nižšie na hrane čierneho kruhu, a najkratšie cesty budú prechádzať touto hranou. To vidno na obrázku v časti (e). Avšak po tréningu dosiahneme, aby najkratšie cesty viedli práve hranou, ktorou chceme (f). Zelená časť už vysegmentovanej časti hranice predstavuje množinu šestnástich pixelov, použitú ako tréningová vzorka. Nová funkcia m'_G, ktorá je výsledkom tréningu, je zobrazená v časti (g). Vidno, že pokiaľ veľkosť gradientu v pixeli nezodpovedá veľkosti gradientu pixelov z tréningovej vzorky, bude mať tento pixel maximálny prírastok k váhe na hrane. Ak má pixel rovnakú veľkosť gradientu, prírastok je nulový. V časti (h) vidno ilustráciu výsledných váh na hranách grafu po tréningu. Čierna farba pixelov reprezentuje najnižšie váhy na hrane. V porovnaní s (b) vidno, že váhy na hrane čierneho kruhu nadobúdajú po tréningu vyššie hodnoty, a naopak, váhy na vonkajšej hrane medzikružia nadobudli nižšie hodnoty. Výsledkom toho je zahrnutie vonkajšej hrany medzikružia do nájdených najkratších ciest.


(a)
(b)
(c)
(d)
f_G
30
25
20
15
10
5
200
400
600
800
1000
(e)
(f)
(g)
(h)
m'_G
30
25
20
15
10
5
200
400
600
800
1000

Obr. 5.8.1

5.9 Prichytávanie kurzoru

Na začiatku segmentácie metódou “Intelligent scissors”je potreba zvoliť počiatočný pixel hranice segmentovanej oblasti. Počas samotnej segmentácie treba voliť pixeli, ktoré tvoria koniec práve segmentovanej časti hranice objektu. Voľba týchto všetkých bodov sa vykonáva umiestnením kurzoru myši nad požadovaný pixel a voľba sa potvrdí kliknutím na myš. Tento spôsob nie je príliš presný z dôvodu malých rozmerov pixelov. Tento problém sa dá vyriešiť takzvaným prichytávaním kurzoru myši. Majme daný pixel, pod ktorým sa práve nachádza kurzor myši. Uvažujeme okolie pixelu, do ktorého zahrnieme všetky pixeli, ktorých vzdialenosť je od daného pixelu menšia alebo rovná určitej vzdialenosti. Po kliknutí na myš bude zvolený pixel, ktorého hodnota veľkosti gradientu je najvyššia spomedzi všetkých pixelov v takto definovanom okolí. V prípade, ak je najvyššia hodnota veľkosti gradientu vo viacerých pixeloch v oblasti, pričom aj v strede oblasti, bude zvolený pixel v strede oblasti.

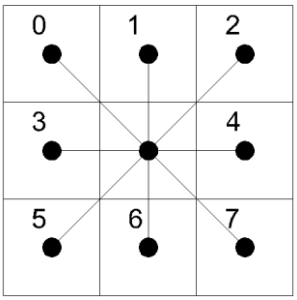
5.10 Technika “boundary cooling”

Technika nazvaná “boundary cooling” umožňuje automatickú voľbu nových pixelov tvoriacich časti segmentovanej hranice. Využíva vlastnosť, že mnohé z najkratších ciest z daného pixelu majú určitú časť cesty spoločnú a zdielajú určité množstvo pixelov. Majme dve najkratšie cesty zo spoločného bodu p do dvoch rôznych bodov. Ak tieto cesty majú spoločný bod q, potom časť najkratšej cesty medzi bodom p a q je pre obe najkratšie cesty totožná. Každý pixel v obraze si uchováva informáciu o dobe, ktorú sa nachádzal vo vykreslenej najkratšej ceste. Do tejto doby sa započítava aj preškálovaná hodnota veľkosti gradientu, aby práve tieto pixeli boli preferované v prípade, že sa pixel nachádza na hrane. Okrem tejto doby si pixel uchováva aj počet, koľko krát sa nachádzal v nejakej zobrazenej najkratšej ceste. Obe tieto hodnoty majú svoju hodnotu dolného a horného prahu. Dolný prah slúži na určenie, či je pixel kandidátom na automatickú voľbu pixelu, horný prah určuje či segmentovaná časť je prijateľná. Prvý pixel, ktorý prekročí oba dolné prahy sa stáva kandidátom na automaticky zvolený pixel, a prvý kandidát na práve segmentovanej časti, ktorá obsahuje pixel prekračujúci oba horné prahy, sa stáva automaticky zvoleným pixelom.

6. Implementácia metódy Intelligent scissors

Naším cieľom bolo implementovať metódu Intelligent scissors. Program sme vytvorili v prostredí Microsoft Visual Studio 2013 v integrovanom vývojovom prostredí Microsoft Visual C++ v programovacom jazyku C++/CLI. Vytvárali sme formulárovú aplikáciu s grafickým používateľským rozhraním (Windows Forms Application) pre operačný systém Windows, využívajúcu platformu .NET (verzia 4.5).

Graf sa často v pamäti počítača uchováva ako matica susednosti [3]. Ak n je počet pixelov v obraze, bola by potrebná matica veľkosti $n \times n$, ktorá by bola riedka (maximálne 8 prvkov v riadku) a zaberala by v pamäti počítača veľa miesta. Štruktúra obrazu nám umožnila zvoliť iný, efektívnejší prístup. Na reprezentáciu grafu obrazu sme vytvorili triedy pre graf, vrchol grafu, hranu a farbu. Objekt triedy graf predstavuje jeden konkrétny graf. Obsahuje informáciu o výške a šírke obrazu a jednorozmerné pole vrcholov, ktorého veľkosť je rovná počtu pixelov. Každý vrchol nesie informáciu o farbe pixelu a osemprvkové pole hrán do susedných vrcholov. Indexovanie hrán v tomto poli je znázornené na obrázku 6.1..



Obr. *6.1*

Objekt triedy hrana má ako dáta identifikačné číslo vrcholu, do ktorého smeruje, a váhu na tejto hrane. Toto identifikačné číslo vrcholu je rovné indexu tohto vrcholu v poli vrcholov. V prípade, ak vrchol nemá susedný vrchol (vrcholy na okraji obrazu), identifikačné číslo je nastavené na hodnotu -1. Uvažujme ktorýkoľvek vrchol. K jeho niektorému susednému vrcholu sa vieme jednoducho dostať cez pole hrán, alebo jednoduchým výpočtom indexu v poli vrcholov. Toto nám umožňuje charakteristická štruktúra grafu obrazu. Tieto triedy obsahujú aj niektoré dodatočné dáta a metódy potrebné pre výpočty a beh programu.

Na výpočet hodnôt funkcií f_Z a f_G má užívateľ možnosť zvoliť až tri hodnoty pre štandardnú odchýlku σ, ktorá určuje veľkosť predvyhľadenia obrazu. Pre konkrétnu hodnotu σ treba vypočítať prislúchajúcu šírku konvolučnej masky. Pri tomto výpočte sa

využíva integrácia funkcie dvoch premenných, ktorá je implementovaná Simpsonovým integračným pravidlom pre dvojrozmerné funkcie.

Hodnoty funkcie f_D a f_Z sa vypočítajú iba raz na začiatku segmentácie po načítaní obrazu, a hodnoty ostatných funkcií je treba pri tréningu vypočítavať vždy nanovo. Preto súčet hodnôt funkcií f_D a f_Z ukladáme do triedy, ktorá má štruktúru grafu. Ak treba vypočítať nové hodnoty váh na hranách grafu, tie sa vždy budú rovnať súčtu týchto dvoch hodnôt funkcií plus prírastku od ostatných funkcií. Takto sa skráti výpočtový čas potrebný pre výpočet nových váh na hranách.

Trieda Intelligent_scissors obsahuje dáta a metódy, ktoré špecifikujú jej fungovanie. Príkladom je napríklad hodnota váh w_Z, w_G, w_D, w_P, w_I a w_O, alebo pole pixelov, ktoré tvoria hranicu segmentácie.

Pri aktívnom tréningu sa pri tvorbe histogramov (5.7.2) využíva klesajúca funkcia váh *w(i)*, ktorá vracia vyššiu hodnotu váhy pre pixely, ktoré sa nachádzajú na začiatku tréningovej množiny. Táto funkcia je implementovaná ako

$$w(i) = -\left(\frac{i}{2 \cdot (t_s + 1)}\right) + 1 \tag{6.1}$$

kde t_s je počet pixelov v tréningovej množine a $i = 0, 1, \ldots, t_s - 1$. Je to lineárna klesajúca funkcia, pre ktorú platí, že $w(0) = 0$ a $w(t_s - 1) = 0{,}5$.

Výpočtová zložitosť implementovaného Dijkstrovho algoritmu je $O(|E| \cdot \log(|V|))$. Prioritná fronta bola použitá z knižnice STL. Na hľadanie najkratších ciest v grafe bola vytvorená samostatná trieda.

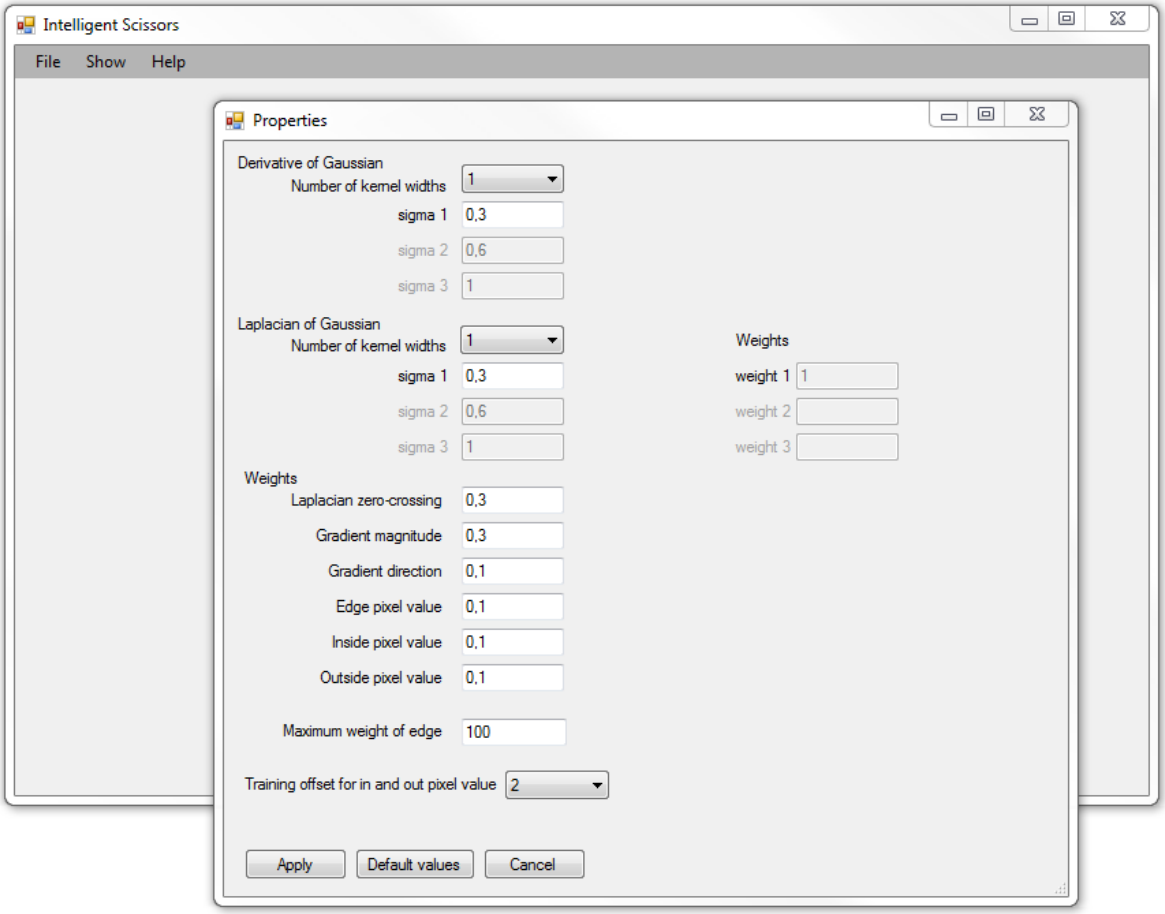
Na konci segmentovania je potrebné hranicu segmentu uzatvoriť zvolením posledného pixelu v mieste počiatočného pixelu. Toto môže byť náročné z dôvodu malých rozmerov pixelov. Tento problém sme vyriešili tým, že užívateľ má možnosť zvoliť veľkosť okolia, ktoré je definované rovnako ako okolie pre prichytávanie kurzoru. Teda, ak sa počiatočný pixel nachádza v okolí naposledy zvoleného pixelu v obraze, segmentačná hrana sa automaticky uzavrie.

Po vytvorení uzavretého segmentu má užívateľ možnosť odstrániť okolie segmentovanej oblasti. Za týmto účelom bolo potrebné implementovať riadkové semienkové vypĺňanie oblasti (“scan-line seed fill”). Takto sme dokázali určiť množinu všetkých pixelov obrazu nachádzajúcich sa vo vysegmentovanej oblasti.

Technika “boundary cooling” nie je v našom programe implementovaná.

6.1 Popis užívateľského prostredia programu

Užívateľské prostredie je veľmi jednoduché a intuitívne. Prvým krokom je načítanie obrazu. Podporované sú formáty JPG, PNG a BMP. Obraz sa načíta kliknutím myšou na ponuku *File→Open*, a následným výberom obrazu zo štandardného okna na otváranie súborov. Po načítaní súboru sa zobrazí okno s nastaveniami (obrázok 6.1.1).



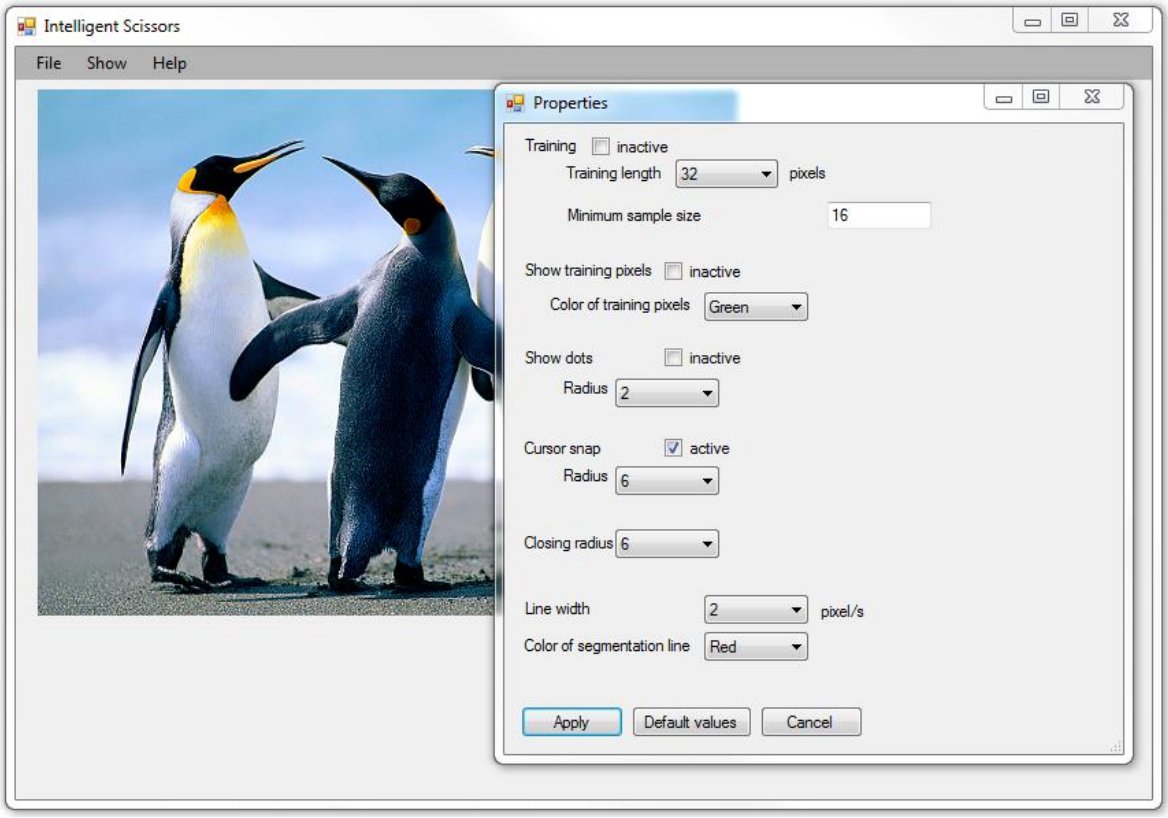
Obr. *6.1.1*

Všetky číselné vstupy v tvare desatinného čísla je potreba zadávať s desatinnou čiarkou. Prvá štvorica nastavení predstavuje počet konvolučných masiek s nastaviteľnou hodnotou štandardných odchýlok pre výpočet hodnôt funkcie veľkosti gradientu f_G. Ďalšia štvorica nastavuje to isté, avšak pre funkciu f_Z. Po pravej strane je nastavenie príslušných váh pre jednotlivé konvolučné jadrá. Šestica váh nastavuje veľkosť prírastku jednotlivých funkcií k váham na hranách grafu (funkcia 5.1.1). Ako posledné je možné nastaviť maximálnu celočíselnú hodnotu váhy na hrane, a hodnotu *k* z funkcií 5.5.2 a 5.5.3. Všetky tieto nastavenie sú prístupné iba pri načítavaní nového obrazu. Po potvrdení tlačidlom *Apply* sa načítaný obraz zobrazí v hlavnom okne programu. Okrem týchto nastavení sú k dispozícii aj ďalšie (obrázok 6.1.2). Prístup k nim je cez ponuku

File→Properties a sú prístupné pred načítaním obrazu, ale aj kedykoľvek počas segmentácie.

Počas segmentácie je možné kedykoľvek aktivovať tréning a nastaviť maximálnu dĺžku tréningovej vzorky, ako aj minimálnu veľkosť tréningovej vzorky, ktorú budeme považovať za dostatočne veľkú na vytvorenie dynamickej funkcie veľkosti gradientu m'_G. Ďalšou možnosťou je zvýraznenie tréningových pixelov, ktoré môžu byť zahrnuté do tréningu. Nastaviteľná je aj ich farba. Toto zobrazenie môže byť aktívne aj v prípade neaktívneho tréningu a slúži pre predstavu užívateľa, ktoré pixeli by tvorili tréningovú vzorku. Metódou "Intelligent scissors" segmentujeme hranicu po častiach.

Možnosť "*Show dots*" slúži na zobrazenie bodov v miestach, kde sa zadal počiatok častí hranice segmentu. Prichytávanie kurzoru a uzatváranie hranice spresňuje a urýchľuje prácu užívateľa (prichytávanie kurzoru sme popísali v kapitole 5.9, uzatváranie hranice čitateľ nájde v kapitole 6). Ak je prichytávanie kurzoru aktívne, kružnica v okolí kurzoru myši naznačuje oblasť, v ktorej dochádza k prichytávaniu. Ako posledné je možné nastaviť hrúbku a farbu zobrazovaných najkratších ciest spolu s doposiaľ vysegmentovanou časťou hranice segmentu.

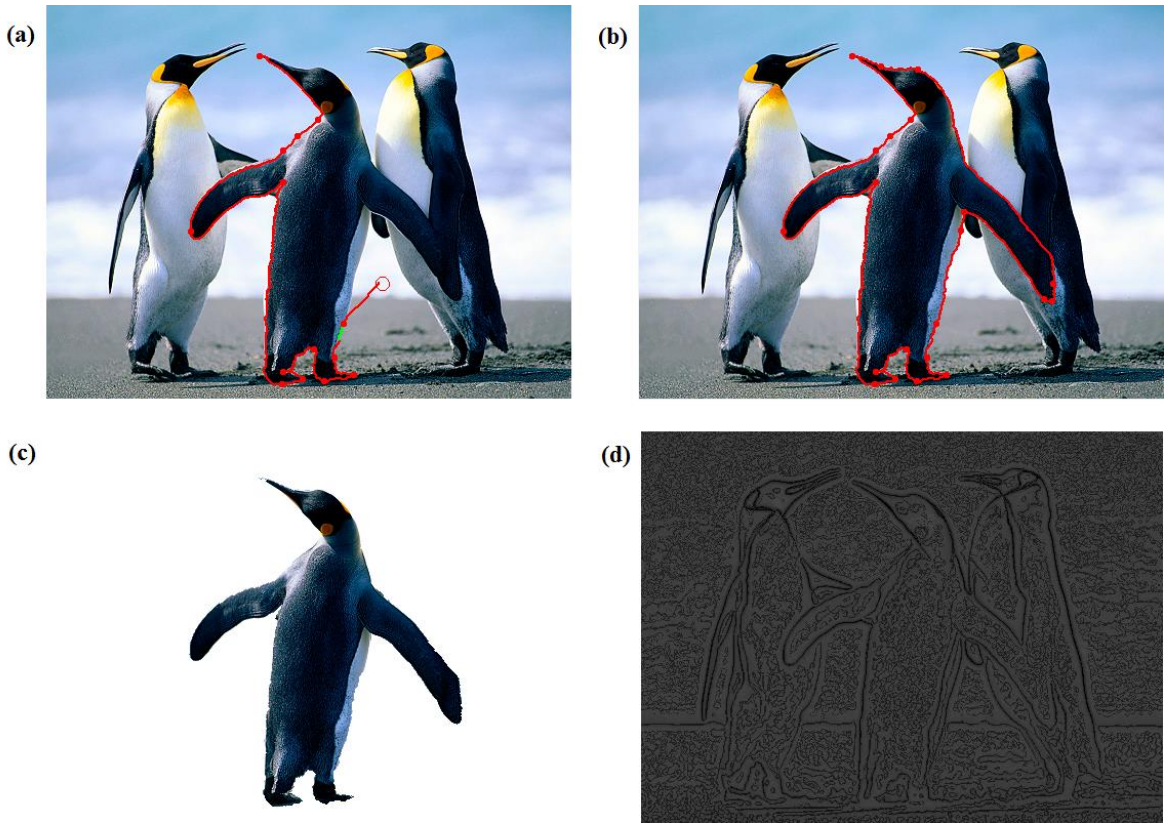


Obr. *6.1.2*

Prvým krokom samotnej segmentácie je zvolenie počiatočného bodu. Z tohto bodu začína vytváranie hranice segmentácie. Užívateľ pohybom kurzoru myši a kliknutím na

myš zvolí nový úsek, ktorý bude tvoriť novú časť segmentovanej hranice. Po každom kliknutí myšou sa takto zvolí vždy nový bod hranice, z ktorého sa vykresľuje najkratšia cesta v grafe po aktuálnu polohu kurzoru myši (obrázok 6.1.3 (a)). Po tom, ako sa segmentovaná hranica uzavrie (obrázok 6.1.3 (b)), je obraz rozdelený hranicou segmentu na dve oblasti. Užívateľ zvolí oblasť, ktorú považuje za výsledný segment kliknutím myši na pixel v danej oblasti. Po tom, ako je oblasť ohraničená a zvolená, si užívateľ môže uložiť do textového súboru súradnice pixelov tvoriacich hranicu segmentu (*File→Save→Coordinates*), alebo obrázok zvolenej vysegmentovanej časti obrazu vo formáte PNG (obrázok 6.1.3 (c)). Túto operáciu vykoná pomocou ponuky *File→Save→Segment*.

Kedykoľvek počas segmentácie si užívateľ môže prezrieť ilustračný náhľad hodnôt váh na hranách grafu pomocou ponuky *Show→Graph weights* (obrázok 6.1.3 (d)). Tento náhľad je ilustračný z dôvodu nemožnosti zobraziť hodnoty ôsmych váh z vrcholu alebo do vrcholu grafu ako dvojrozmerný obrázok. Preto bola vykreslená iba hodnota jednej hrany, konkrétne váhy k ľavému hornému susedovi.



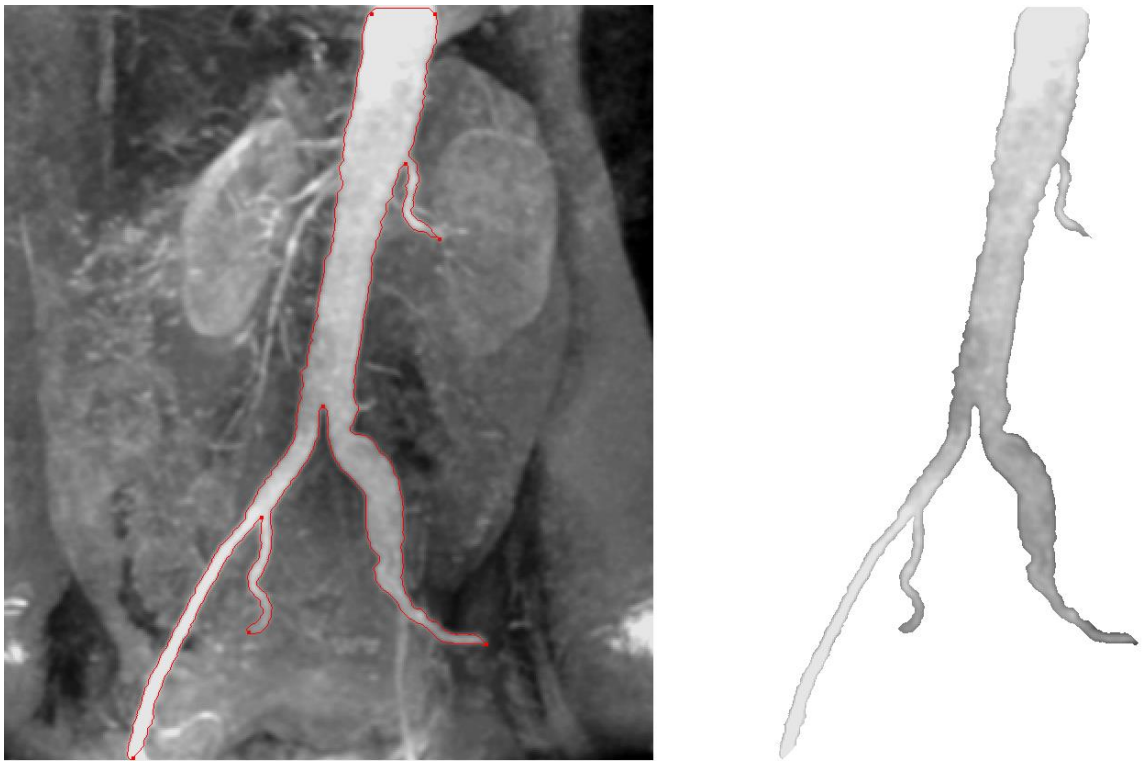
Obr. *6.1.3*

Aplikáciu užívateľ môže kedykoľvek ukončiť zvolením ponuky *File→Close*. Základné informácie o programe sa nachádzajú v ponuke *Help→About*.

7. Aplikačná časť

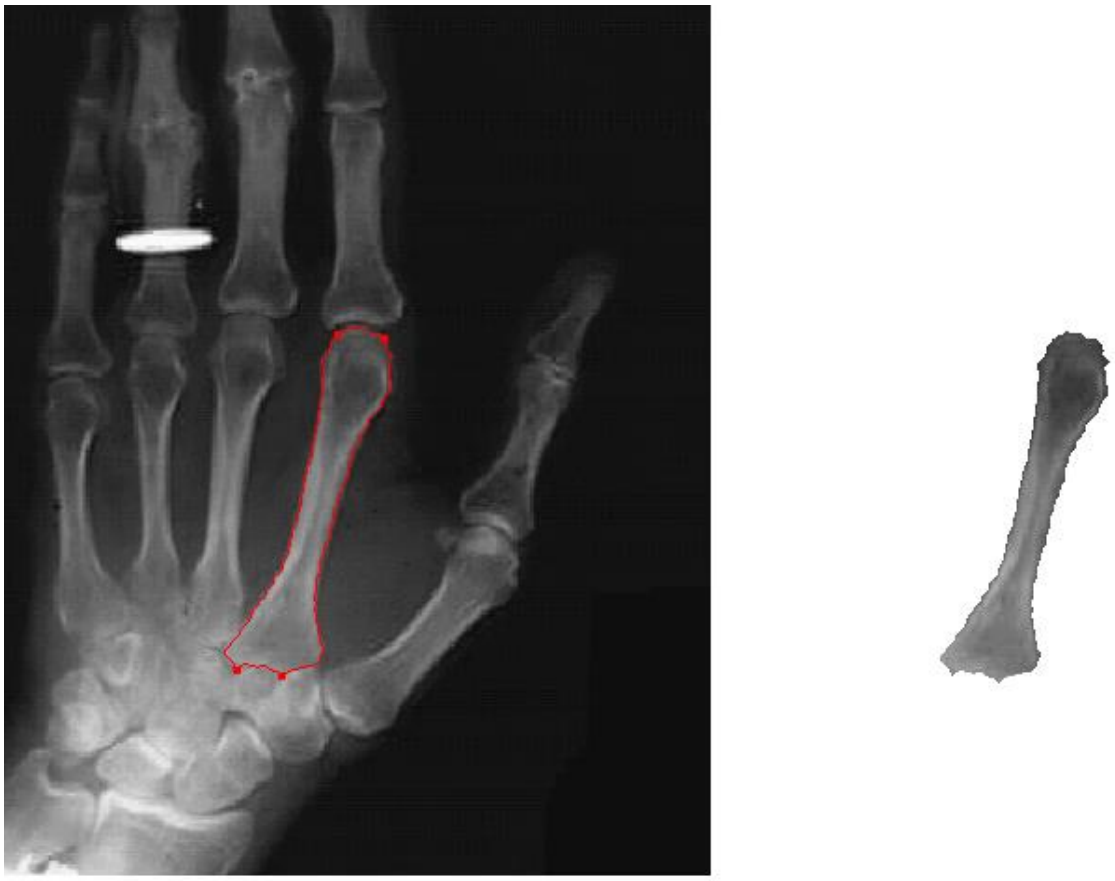
V tejto časti práce otestujeme nami implementovanú segmentačnú metódy "Intelligent scissors" na niekoľkých obrazoch.

Prvým použitým obrazom je medicínsky snímok obličky (obrázok 7.1). Na výpočet výslednej funkcie veľkosti gradientu boli použité dve konvolučné masky so štandardnou odchýlkou $\sigma_1 = 0,3$ a $\sigma_2 = 0,6$. Voľba nízkych hodnôt bola z dôvodu, aby sa hrany obrazu príliš nerozmazali. Výslednú funkcie f_Z sme taktiež počítali z dvoch rôznych masiek, kde $\sigma_1 = 1,5$ a $\sigma_2 = 3$. V tomto prípade sem použili naopak vyššie hodnoty. Tie spôsobia nájdenie menšieho počtu hrán funkciou f_Z, pričom slabšie hrany zaniknú a silnejšie pretrvajú. Maximálnu hodnotu váhy na hrane volíme 50. Táto hodnota je postačujúca, pričom vyššie hodnoty spôsobujú dlhší výpočtový čas Dijkstrovho algoritmu.



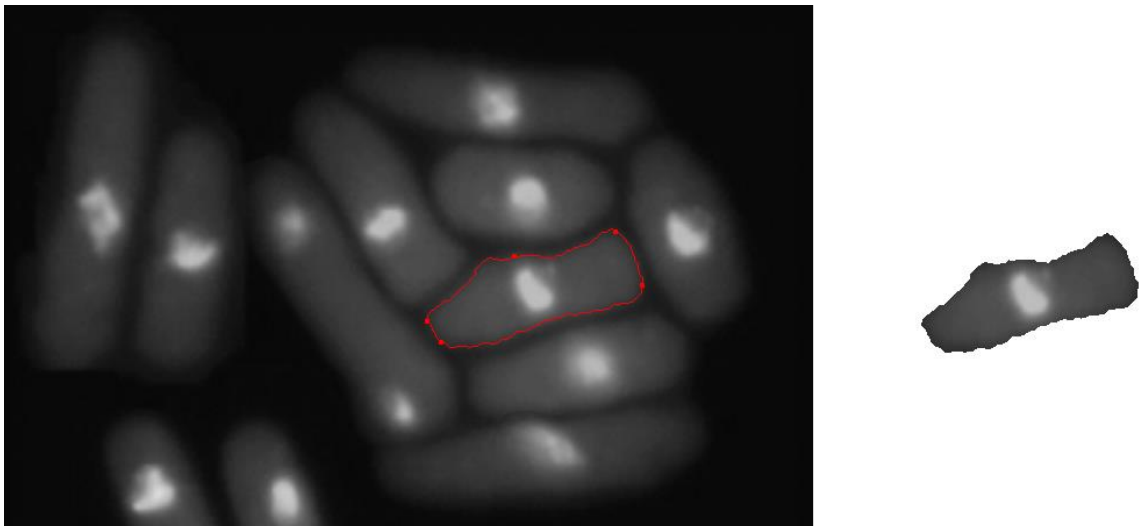
Obr. *7.1*

Druhým použitým obrazom bola röntgenová snímka ľavej ruky (obrázok 7.2). Hodnoty σ pre výpočet výslednej funkcie veľkosti gradientu : $\sigma_1 = 0,3$ a $\sigma_2 = 0,6$. Hodnota σ pre funkciu f_Z bola zvolená $\sigma = 2$.



Obr. *7.2*

Predposledným zvoleným obrazom bol mikroskopický snímok kvasníc (obrázok 7.3). Vhodné hodnota pre výpočet funkcie veľkosti gradientu bola $\sigma = 0{,}3$. Na výpočet funkcie f_Z sme použili rovnaké hodnoty ako pri obrázku 7.1, $\sigma_1 = 1.5$ a $\sigma_2 = 3$.



Obr. *7.3*

Posledným zvoleným obrazom bol snímok tučniakov (obrázok 7.4). Hodnoty σ pre výpočet výslednej funkcie veľkosti gradientu boli zvolené ako $\sigma_1 = 0{,}3$, $\sigma_2 = 1$ a $\sigma_3 = 1{,}5$. Pre funkciu f_Z sme tieto hodnoty zvolili $\sigma_1 = 1$, $\sigma_2 = 2$ a $\sigma_3 = 3$;

Obr. *7.3*

8. Záver

Metóda “Intelligent scissors” má ako každá iná metóda svoje výhody aj nevýhody. Hlavnou nevýhodou metódy je dĺžka výpočtového času Dijkstrovho algoritmu. Avšak po implementácii efektívnejšej prioritnej fronty by mohol tento problém zmierniť. Ďalšou nevýhodou je nutnosť poznať fungovanie metódy, alebo prinajmenšom vedieť, čo má za následok zmena konkrétneho parametra. Hlavnú výhodu tejto metódy vidíme v tréningu, ktorý umožňuje metóde adaptovať sa na konkrétny druh hrany, ktorú potrebujeme segmentovať. Ďalšou výhodou je množstvo nastavení, ktoré umožňujú užívateľovi prispôsobiť metódu na široké množstvo rôznych obrazov. Nezanedbateľný vplyv má aj účasť užívateľa v procese segmentácie, čo výrazne napomáha k dosiahnutiu kvalitných výsledkov.

Nami implementovaný program ponúka priestor pre jeho ďalšie vylepšenie. Navrhovanými vylepšeniami sú napríklad efektívnejšia implementácia Dijkstrovho algoritmu, alebo možnosť segmentovať viac segmentov zároveň. Náš program umožňuje uložiť do súboru súradnice pixelov tvoriacich segment. Užitočná by mohla byť spätná možnosť, kedy by sa súradnice nahrali späť do programu a segmentácia by mohla ďalej pokračovať. Užitočným by určite bola aj možnosť vrátiť sa o jeden krok späť.

Zoznam bibliografických odkazov

1. PENG, B., L. ZHANG a D. ZHANG: A survey of graph theoretical approaches to image segmentation. Pattern recognition. Elsevier 2013, roč. 46, č. 3, s. 1020-1038. ISSN 0031-3203.

2. SONKA, M., V. HLAVAC a R. BOYLE: Image processing, analysis, and machine vision. Springer US 2008, ISBN 0-495-24438-4.

3. KNOR, M.: Teória grafov. Vydavateľstvo STU, 2008, Bratislava. ISBN 978-80-227-2879-9.

4. K. MAGZHAN a H. MAT JANI: A review and evaluations of shortest path algorithms. International journal of scientific & technology research. 2013, roč. 2, č. 6. ISSN 2277-8616.

5. CORMEN, T., LEISERSON, CH., R. RIVEST a C. STEIN: Introduction to algorithms. The MIT Press, 2009. ISBN 978-0-262-03384-8

6. E. MORTENSEN a W. BARRETT: Interactive segmentation with Inteligent scissors. Graphical models and image processing. Brigham Young University 1998, roč. 60, č. 5, s. 349-384. ISSN 1077-3169