UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

POKROČILÉ METÓDY DETEKCIE KONTÚR A ICH POUŽITIE PRI ROZPOZNÁVANÍ TRASOLOGICKÝCH OBJEKTOV

Diplomová práca

dc677e0f-e970-4b8a-8ed6-e0fb83126588

2017 **Bc. Marcel Koč**

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

POKROČILÉ METÓDY DETEKCIE KONTÚR A ICH POUŽITIE PRI ROZPOZNÁVANÍ TRASOLOGICKÝCH OBJEKTOV

Diplomová práca

dc677e0f-e970-4b8a-8ed6-e0fb83126588

Študijný program: Aplikovaná informatika

Študijný odbor: 9.2.9 Aplikovaná informatika

Katedra: KIN FPV – Katedra informatiky

Vedúci diplomovej práce: RNDr. Alžbeta Michalíková, PhD.

Banská Bystrica, 2017 **Bc. Marcel Koč**

Univerzita Mateja Bela v Banskej Bystrici
Fakulta prírodných vied

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta:	Bc. Marcel Koč
Študijný program:	aplikovaná informatika (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor:	aplikovaná informatika
Typ záverečnej práce:	Magisterská záverečná práca
Jazyk záverečnej práce:	slovenský
Sekundárny jazyk:	anglický

Názov: Pokročilé metódy detekcie kontúr a ich použitie pri rozpoznávaní trasologických objektov

Anotácia: Na katedre informatiky toho času prebieha riešenie projektu APVV pod názvom „Automatizované spracovávanie trasologických objektov". Uveďte a stručne opíšte základné spôsoby detekcie kontúr v obraze. Vytvorte aplikáciu, ktorá bude pracovať s trasologickými objektami a pomocou aktívnych kontúr vyhľadá objekty v obraze. Porovnajte získané výsledky s výsledkami získanými použitím iných metód a navrhnite možnosti zlepšenia, respektíve kombinácie jednotlivých metód. K vytvorenému programu vypracujte používateľskú príručku a systémovú dokumentáciu.

Vedúci:	RNDr. Alžbeta Michalíková, PhD.
Katedra:	KIN FPV - Katedra informatiky
Vedúci katedry:	RNDr. Miroslav Melicherčík, PhD.
Dátum zadania:	07.04.2016
Dátum schválenia:	10.04.2017

prof. RNDr. Roman Nedela, DrSc.
garant študijného programu

Čestné vyhlásenie

Čestne vyhlasujem, že som túto diplomovú prácu vypracoval samostatne s použitím uvedených zdrojov.

V Banskej Bystrici, dňa 26.4.2017 .....

Poďakovanie

Touto cestou chcem poďakovať RNDr. Alžbete Michalíkovej, PhD., vedúcej mojej diplomovej práce a taktiež Mgr. Michalovi Vagačovi, PhD., za pomoc, čas a inšpiráciu pri písaní práce.

Abstrakt

KOČ, Marcel: Pokročilé metódy detekcie kontúr a ich použitie pri rozpoznávaní trasologických objektov. [Diplomová práca] / Marcel Kočˇ. - Univerzita Mateja Bela v Banskej Bystrici. Fakulta prírodných vied; Katedra informatiky. - Školiteľ: RNDr. Alžbeta Michalíková, PhD. Banská Bystrica FPV UMB, 2017, 62 s.

Cieľom práce je implementovať algoritmy aktívnych kontúr určených na segmentáciu objektov v obraze, otestovať a porovnať tieto algoritmy medzi sebou. Algoritmy boli testované na segmentácií trasologických objektov, pričom test bol zameraný hlavne na výpočtový čas a kvalitu výsledku. Záverom tejto práce sú navrhnuté vylepšenia jednotlivých implementácií.

Kľúčové slová: Snakes, Level Sets, Fast Marching, aktívne kontúry, trasologické objekty.

Abstract

KOČ, Marcel: Advanced techniques of contour detection and their application for recognition of trasology object. [Master thesis] / Marcel Koč. - Matej Bel University Banská Bystrica. Faculty of Natural Sciences; Department of computer sciences. - Supervisor: RNDr. Alžbeta Michalíková, PhD. Banská Bystrica FPV UMB, 2017, 62 p.

The main goal of this thesis is to implement active contours algorithms for segmenting traceological objects in images, and to test and compare those algorithms. The tests focus on computing time and quality of results. To conclude, enhancements to individual algorithm implementations are discussed.

Keywords: Snakes, Level Sets, Fast Marching, active contours, trasological objects.

Predhovor

Každým dňom je vytvorených čoraz viac a viac obrazových dát, čo je spôsobené rozšírením technológií do všetkých vrstiev spoločnosti. Tieto dáta je potrebné spracovať, analyzovať a uschovať. Avšak niektoré zariadenia nám môžu poskytnúť citlivé dáta, ktorých analýzou môžeme napríklad zachrániť ľudský život. Množstvo odvetví používajúce analýzu obrazu v reálnom čase pribúda, ako napríklad medicína, robotika s podobne.

Úlohou tejto práce je vytvoriť aplikáciu, v ktorej sú implementované algoritmy na spracovanie obrazu, konkrétne pomocou metód aktívnych kontúr. Vytvorená aplikácia poslúži pri segmentácií trasologických objektov v obraze. Okrem iného, tak prispeje práve prebiehajúcemu vedecko-výskumnému projektu Katedry informatiky, ktorý vznikol prostredníctvom grantovej schémy Agentúry na podporu výskumu a vývoja. Projekt je priamo orientovaný do oblasti hospodárskej a spoločenskej praxe, pričom odberateľom výstupov a výsledkov projektu je Policajný zbor Slovenskej republiky. Samotná práca implementuje a porovnáva algoritmy aktívnych kontúr, ktoré sú otestované na obrázkoch trasologických objektov. Výsledkom práce je hotová aplikácia, ktorá môže slúžiť k zrýchleniu a zefektívneniu práce kriminalistických expertov pri detekcií a rozpoznávaní stôp z miesta činu.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Zoznam skratiek a značiek

Úvod

V posledných rokoch dochádza k neustálemu vývoju a rozširovaniu technológií do širokého spektra verejnosti, čo spôsobuje rapídny nárast objemu dát, ktoré je potrebné spracovať, vyhodnocovať a uchovávať. Špecialisti preto hľadajú spôsoby, aby postupy na dosiahnutie týchto cieľov boli dostatočne presné a čo najefektívnejšie. Táto práca s názvom „Pokročilé metódy detekcie kontúr a ich použitie pri rozpoznávaní trasologických objektov"je zameraná na implementáciu a porovnanie algoritmov aktívnych kontúr, ktoré slúžia na segmentáciu objektov v obraze. Trasologickými objektmi rozumieme odtlačky pneumatík, dezény kolies, časťami tiel, oblečením a podobne. Inými slovami, tieto objekty sú získavané kriminalistickými expertmi z miesta činu. Spracovaním týchto objektov môžeme pomôcť kriminalistickým expertom určiť podmienky vzniku týchto stôp, poprípade určiť prítomnosť osôb a predmetov na mieste činu.

Diplomová práca svojim zameraním nadväzuje na práve prebiehajúci výskumný projekt s názvom *„Automatizované spracovávanie trasologických objektov"*. Projekt je realizovaný Katedrou informatiky Fakulty prírodných vied Univerzity Mateja Bela v Banskej Bystrici a financovaný Ministerstvom školstva, vedy, výskumu a športu Slovenskej republiky, prostredníctvom grantovej schémy Agentúry na podporu výskumu a vývoja s kódom APVV-0219-12. Hlavným cieľom je implementovať algoritmy aktívnych kontúr, otestovať a porovnať jednotlivé implementácie a navrhnúť vylepšenia. V našom prípade je program určený na segmentáciu trasologických objektov v obraze, konkrétne na nájdenie dezénu pneumatík.

Vnútorná štruktúra dokumentu je rozdelená na 9 kapitol. Prvé štyri kapitoly sú v teoretickej rovine, zvyšné kapitoly sú praktické. V prvej kapitole je stručne uvedené, akým spôsobom je obraz reprezentovaný v počítači, a aké druhy obrazu sme v našej práci použili. V nasledujúcej kapitole sa zaoberáme segmentáciou obrazu, jej významom, použitím a opisom základných metód spolu s grafickou ukážkou výsledkov. Postupne sa dostaneme k teoretickému jadru zamerania tejto práce a v tretej kapitole uvedieme čitateľa do problematiky aktívnych kontúr. V tejto kapitole je stručný popis rozdelenia aktívnych kontúr a vysvetlenie jednotlivých princípov. Vo štvrtej kapitole sa oboznámime so spôsobom, ako náš program je navrhnutý. Vysvetlíme si princíp prepojenia jednotlivých komponentov a pripravíme sa tak na detaily ohľadom implementácií jednotlivých algoritmov.

Po tejto kapitole nasledujú tri kapitoly, z ktorých sa každá venuje jednému algoritmu. Piata kapitola je zameraná na Snakes algoritmus, ktorý patrí do kategórie implicitných aktívnych kontúr. V tejto kapitole sa dočítame, kedy algoritmus vznikol, kto je jeho tvorcom, matematické vyjadrenie algoritmu, princíp fungovania a podobne. Taktiež si tu ukážeme grafické ukážky priebehu výpočtu, a aké výhody a nevýhody nám tento algoritmus ponúka. V ďalšej kapitole je prezentovaný algoritmus Level Sets, ktorý je zástupcom kategórie implicitných aktívnych kontúr. Rovnako ako pri predošlom algoritme, aj tu sa dočítame, kto je autorom algoritmu, potrebnú teóriu a princíp fungovania implementácie. Ukážeme si grafickú ukážku priebehu a poukážeme na výhody a nevýhody algoritmu. Postupne tak prejdeme do ďalšej kapitoly, ktorá obsahuje Fast Marching algoritmus, ktorý rovnako ako Level Sets patrí do kategórie implicitných aktívnych kontúr. Aj v tomto prípade sa dočítame, kto je považovaný za autora algoritmu, na akom princípe algoritmus funguje. Taktiež si ukážeme, ako funguje algoritmus a nezabudneme ani na výhody a nevýhody algoritmu. Každá z kapitol opisujúcich algoritmus obsahuje ešte implementačné detaily. V týchto detailoch je možné nájsť parametre potrebné pre fungovanie algoritmu, a taktiež spôsob, akým sú tieto algoritmy implementované, a akým spôsobom implementácie fungujú.

Po vysvetlení a ukážke jednotlivých implementovaných algoritmov sa dostaneme k ich použitiu na segmentovanie trasologických objektoch. Vysvetlíme si, ako sme algoritmy testovali, aké výsledky sme z testu získali a vyvodíme záver. Nakoniec navrhneme vylepšenia nášho programu, a tým uzavrieme túto prácu.

Pri spracovávaní témy sme prevažne používali zahraničné zdroje vo forme odborných článkov z internetových zdrojov. Je potrebné konštatovať, že v Slovenskej Republike sa tejto problematike nevenuje veľká pozornosť, a preto je takmer nemožné nájsť odborné knižné publikácie.

1 Digitálny obraz

Digitálny obraz je možné získať množstvom spôsobov, napríklad z kamerových systémov, biomedicínskych prístrojov a podobne. Niekedy je potrebné spracovávať obraz v reálnom čase, ako to je napríklad pri autonómnych vozidlách, ktoré sú ovládané pomocou systému kamier. Na základe vyhodnocovania spracovaného obrazu sa vozidlo pohybuje určitým smerom. Tento príklad je jednoduchou ukážkou, prečo je potrebné, aby algoritmy a spôsoby na spracovanie obrazu boli jednoduché a výpočtovo nenáročné. V našej práci sa budeme zaoberať niekoľkými algoritmami určených na segmentáciu obrazu.

1.1 Reprezentácia digitálneho obrazu

Digitálny obraz je uchovávaný ako matica bodov, ktorých hodnota reprezentuje farbu podľa daného farebného modelu. Rovinný obraz (2D) je reprezentovaný dvojrozmernou maticou, ktorej jednotlivé body nazývame pixely. Priestorový obraz (3D) je naopak reprezentovaný trojrozmernou maticou a jej body nazývame voxely. V našej implementácii sme použili tri druhy reprezentácie obrazu a to:

- **Farebný obraz** - je tvorený trojicou hodnôt reprezentujúcich farby farebného modelu *RGB*. Tento model je tvorený zo základných farieb červenej (angl. *„red"*), zelenej (angl. *„green"*) a modrej (angl. *„blue"*).
- **Čiernobiely obraz** - je reprezentovaný jednou hodnotou v rozsahu 0-255, pričom 0 reprezentuje čiernu farbu a 255 naopak bielu. Všetko ostatné v tomto rozsahu je určitým odtieňom šedej.
- **Binárny obraz** - je reprezentovaný dvoma hodnotami a to 0 alebo 1, pričom 0 reprezentuje čiernu farbu a 1 naopak bielu.

2 Segmentácia obrazu

Segmentácia obrazu je veľkou výzvou oblasti digitálneho spracovania obrazu. Hlavným cieľom segmentácie je rozdeliť vstupný obraz na niekoľko oblastí, ktoré zdieľajú spoločné vlastnosti ako je farba, intenzita farby, textúra a pod. Inými slovami, snažíme sa pixely (alebo voxely v prípade 3D obrazu) v popredí (reprezentujúce objekt) oddeliť od pixelov (alebo voxelov) v pozadí.

Segmentácia je jedným zo základných stavebných kameňov analýzy digitálnych obrazov. Je to zložitý a mnohokrát aj výpočtovo náročný proces, ktorý je ovplyvnený množstvom faktorov ako kvalita vstupu, kvalita zariadenia (ktorým bol vstup zaznamenaný) a podmienky pri ktorých bol vstup zaznamenaný (svetelné podmienky, prostredie a pod.)

Segmentácia je veľmi využívanou metódou pri spracovávaní obrazu a svedčí o tom aj rozsah uplatnenia. Využíva sa v rôznych oblastiach od riadenia autonómnych vozidiel cez počítačové videnie až po detekciu objektov v medicínskych snímkach.

Aj napriek tomu, že segmentácia je oblasťou, ktorá je tu už dlhé roky, stále neexistuje univerzálny algoritmus. Segmentácia je doménovo špecifický problém, následkom čoho je potrebné algoritmy a prístupy upravovať jednotlivým problematikám. Pri segmentácií objektov z obrazu sa dosť často využíva kombinácia viacerých metód, za účelom získania čo najpresnejších výsledkov. My si vymenujeme rozdelenie základných metód, ktoré si jednoducho popíšeme.

Rozdelenie segmentačných techník:

- aktívne kontúry,
- prahové metódy,
- metódy založené na detekcii hrán,
- regionálne metódy,
- metódy zhlukovania,
- ostatné metódy.

2.1 Prahové metódy

Prahové metódy (angl. *„Thresholding"*) sú najjednoduchším spôsobom segmentácie, ktorý delí body obrazu do dvoch skupín na základe prahovej hodnoty (angl. *„threshold"*). Zaradenie do skupiny závisí od toho, či je hodnota bodu väčšia alebo menšia ako prahová hodnota. Najčastejšie používanou vlastnosťou používanou ako prahová hodnota je jas.

Rozlišujeme dva základné druhy prahových metód a to:

- **Globálne prahovanie** - prahová hodnota je určená pre celý obraz.
- **Lokálne prahovanie** - prahová hodnota sa určuje na základe hodnôt v okolí skúmaného bodu.

Veľkou výhodou prahových metód je jednoduchosť, čo má za následok výpočtovú a časovú nenáročnosť. Na druhej strane, jednoduchosť je aj najväčšou nevýhodou, keďže uplatnenie nájde len pri jednoduchých segmentáciách, zložitejšie sú problematické. [1]

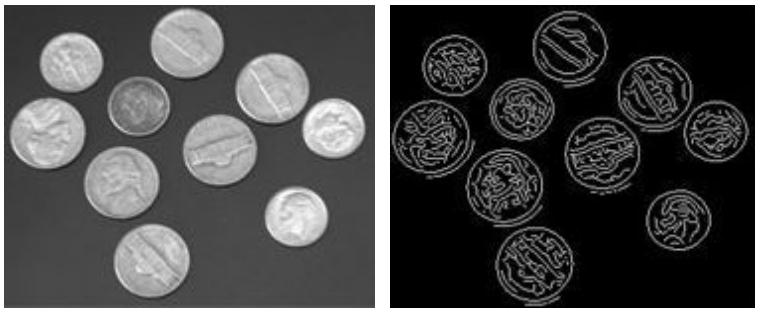
(a) (b)

Obr. 1: Ukážka prahovej segmentácie. *(Zdroj: [2])*

2.2 Metódy založené na detekcii hrán

Metódy založené na detekcii hrán, ktoré na začiatok potrebujú hranový obraz. Hrany v obraze sú reprezentované ako veľké zmeny v hodnote jasu. Hranovým obrazom teda rozumieme množinu bodov, kde prudko narastá alebo klesá jas. Na vytvorenie hranového obrazu metóda potrebuje získať tieto body, a to za použitia prvej alebo druhej derivácie intenzity jasu. Na takomto princípe fungujú:

- Robertsov operátor.
- Prewittov operátor.
- Sobelov operátor.
- Laplacov operátor.



(a) (b)

Obr. 2: Detekcia hrán použitím Sobelovho operátora. *(Zdroj: [3])*

Do tejto oblasti patrí aj metóda aktívnych kontúr, a práve táto metóda je hlavným zameraním našej práce. Teórii aktívnych kontúr sa však budeme venovať v samostatnej kapitole. [1]

2.3 Metódy založené na detekcii oblastí

Metódy založené na detekcii oblastí rozkladajú obraz na množinu oblastí (regiónov), pričom každá oblasť reprezentuje určitú vlastnosť alebo objekt.

Prvou metódou v tejto triede je metóda rozrastajúcich sa oblastí (angl.: *„Growing regions"*). Na začiatku sa definujú body, ktoré spĺňajú požadované kritéria. Následne sa skúma okolie týchto bodov a pokiaľ okolité body spĺňajú požiadavky, sú priradené k oblasti. Je veľmi dôležité vhodne zvoliť počiatočné body, v opačnom prípade výsledok nemusí byť zhodný s očakávaniami.

Druhou metódou je metóda rozdeľ a spoj (angl.: *„Split and merge"*), ktorá funguje presne naopak. Začína sa s celým obrazom, ktorý je rekurzívne rozdelený, väčšinou na štyri podoblastí. Delenie sa zastaví ak oblasť spĺňa požadovanú úroveň homogenity. Výsledkom takého delenia je množina oblastí rôznej veľkosti. Môže nastať situácia, kedy sa počas

priebehu výpočtu rozdelia aj homogénne oblasti. Preto sa v druhom kroku začne následné spájanie susedných oblastí s rovnakou úrovňou homogenity. [1]

(a) (b)

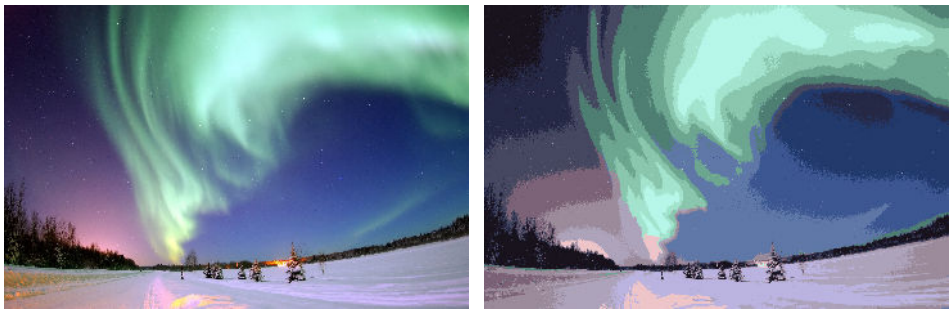
Obr. 3: Ukážka segmentácie detekciou oblastí. *(Zdroj: [4])*

2.4 Metódy zhlukovania

Metódy zhlukovania riešia segmentáciu ako problém klasifikácie. Každý obraz je množinou určitého počtu zhlukov a každý bod (pixel) patrí práve do jedného zhluku. Každý takýto zhluk bodov reprezentuje určitú vlastnosť alebo objekt na obraze.

Prvou metódou je *„K-Means"*, ktorá je všeobecne známa pri riešení klasifikačných problémov. Na začiatok sa určia centrá zhlukov a následne sa prechádza množina bodov obrazu, pričom každý bod sa priradí práve jednému zhluku. Priradenie bodov do zhlukov je definované splnením požiadavky najväčšej podobnosti so zhlukom. V ďalšom kroku sa vypočítavajú nové centrá zhlukov. Výpočet končí, pokiaľ sa všetký centrá zhlukov neprestanú posúvať, respektíve posun nepresiahne určitú hodnotu. Veľkou výhodou je časová efektívnosť, ktorá však nezaručuje aj kvalitu výsledku. Kvalita výsledku závisí od vhodného počatočného zvolenia centier zhlukov.

Ďalšia metóda je označovaná ako *„Mean-Shift"*. Aj táto metóda pracuje so zhlukmi bodov, avšak na začiatok sa zvolia veľkosti vyhľadávacích okien a ich pozície v obraze. Určíme strednú hodnotu sledovanej veličiny (jas, farba a pod.) pre každé z vyhľadávacích okien. Následne každé z vyhľadávacích okien posunie svoje centrum na bod strednej hodnoty. Toto sa opakuje pre každé okno do momentu, kým ich pozícia nezkonverguje. V tomto prípade sa spoja zhluky, ktorých konečná pozícia je rovnaká. Každé vyhľadávacie okno opisuje jednu vlastnosť alebo objekt. [1]



(a) (b)

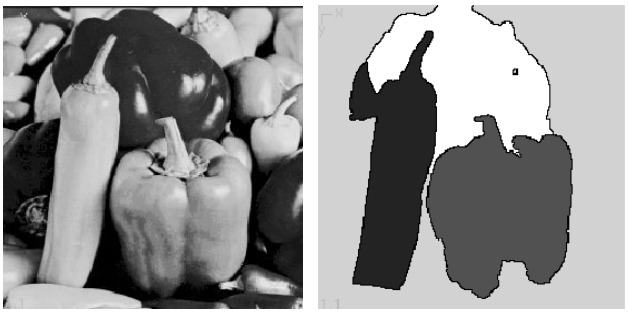
Obr. 4: Ukážka segmentácie K-means algoritmu so 16 zhlukmi. *(Zdroj: [5])*

2.5 Ostatné metódy

Do tejto kategórie patria metódy, ktoré sa nedajú zaradiť do žiadnej z kategórií, lebo kombinujú prístupy viacerých metód a stávajú sa tak z nich hybridné modely.

„Watershed" segmentácia je prvou metódou. Pri tejto segmentácii sa chápe obraz ako množina kopcov v údolí, kde hustota jasu určuje nadmorskú výšku. Segmentáciu si môžeme predstaviť, ako keby sa do údolia vyliala voda, ktorá začne lemovať kopce, a tým segmentuje ich tvar. Táto metóda je náročná ak požadujeme presné výsledky, avšak často dochádza k takzvanému „presegmentovaniu". To v praxi vyzerá tak, že sa „voda rozleje do viacerých údolí" a spájajú sa tak rozdielne oblastí. To je dôvod, prečo treba obraz pred segmentáciou ešte predspracovať.

Medzi hybridné modely patrí aj metóda segmentácie na základe šablóny (angl. *„Template segmentation"*). Túto metódu môžeme použiť, ak hľadáme špecifické vzory alebo poznáme tvar objektu. Metóda vyhodnocuje kritéria pre každú polohu a otočenie vzoru objektu v obraze. Táto metóda je veľmi časovo aj výpočtovo náročná, preto sa jej použitie odporúča len na overenie pár pozícií, ktoré boli získané inou metódou. [1]



(a) (b)

Obr. 5: Segmentácia použitím Watershed algoritmu. *(Zdroj: [6])*

3 Úvod do aktívnych kontúr

V tejto kapitole sa oboznámime s rozdelením metód aktívnych kontúr a vysvetlíme si ako fungujú. Následne s týmito informáciami budeme pokračovať v ďalších kapitolách, kde sa budeme zaoberať implementáciou týchto algoritmov.

Princípom aktívnych kontúr je deformácia počiatočného geometrického modelu, ktorý je reprezentovaný kontúrou (krivkou) v rovine alebo povrchom v trojrozmernom obraze. Pomocou tejto deformácie získavame konečný model rovnakej dimenzie. Celý proces vychádza z vhodne zvolenej funkcie energie. Kontúra sa pohybuje pôsobením vnútorných a vonkajších síl v čase a zastaví sa v mieste s najmenšou energiou. Z toho vyplýva, že princípom aktívnych kontúr je minimalizácia energie.

Funkcia energie má dve základné zložky:

- **Vnútorné energie** - sú závislé na aktuálnom stave modelu a udržiavajú ho v jednom celku.
- **Vonkajšie energie** - sú závislé na čase a umiestnení modelu v obraze. Ich cieľom je priviesť model k objektom záujmu.

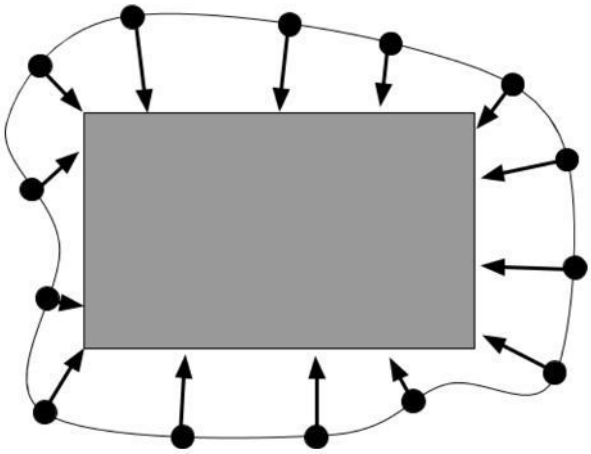
Podľa spôsobu akým sa kontúra pohybuje po obraze, delíme aktívne kontúry na:

- explicitné,
- implicitné.

3.1 Explicitné aktívne kontúry

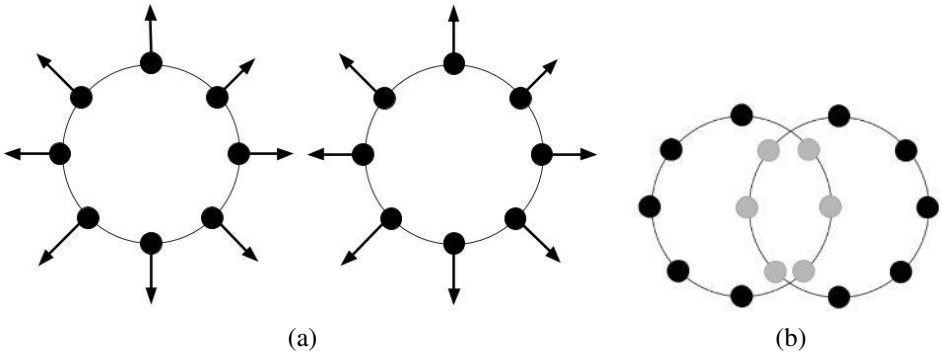
Explicitné aktívne kontúry, inak tiež označované aj ako *„parametrické"*, opisujú kontúru ako množinu bodov, čo môžeme vidieť na obrázku 6. Ako prebieha výpočet, tieto body sa postupne približujú k hranie objektu. Tento prístup je časovo a výpočtovo pomerne nenáročný. Množstvo metód ponúka uspokojivé výsledky avšak len v prípade, že segmentácia prebieha v 2D obraze. Rozšírenie do tretej dimenzie nie je triviálnym problémom vzhľadom k použitiu parametrického vyjadrenia a následného použitia funkcie pre pohyb. Avšak

rozšírenie do tretej dimenzie nie je nemožné, ale zložitosť implementácie nepríjemne vzrastie.



Obr. 6: Ukážka explicitnej kontúry. *(Zdroj: Autor)*

Ďalším nedostatkom tohto prístupu je zmena topológie kontúry. Pohybom v čase sa oblasti, ktoré sú navzájom oddelené, môžu spájať, rozdeľovať a dokonca aj zanikať. Zmeny tejto topológie môžeme vidieť na obrázku 7. Preto je potrebné pri výpočte používať netriviálne prístupy, ktoré vyhodnotia a zaručia potrebné zmeny v topológii kontúry. Taktiež je veľmi obtiažne, a takmer nemožné zistiť, či sa daný bod nachádza v objekte alebo mimo objektu.



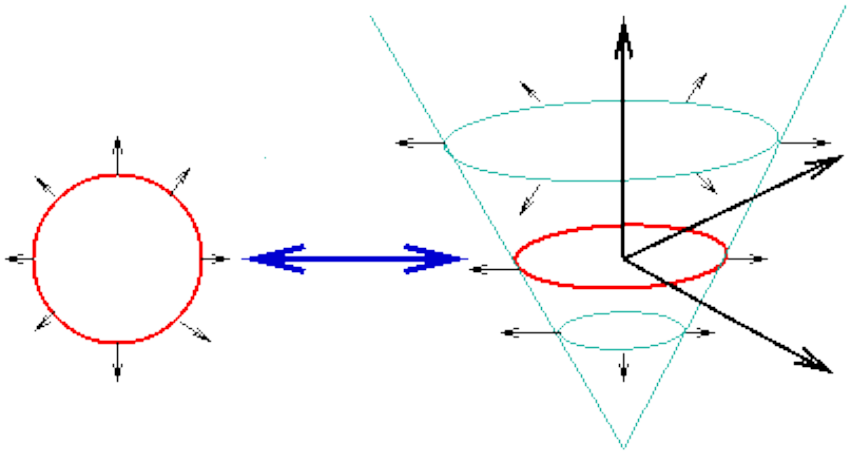
Obr. 7: Ukážka zmeny topológie explicitnej kontúry. *(Zdroj: Autor)*

3.2 Implicitné aktívne kontúry

Implicitné kontúry, taktiež označované aj ako *„geometrické"*, nereprezentujú kontúru ako množinu bodov, ale ako nultú vrstevnicu implicitne zadanej funkcie. Ako môžeme vidieť na obrázku 8, nultá vrstevnica (označená červenou farbou) reprezentuje objekt.

Všetko, čo sa nachádza pod nultou vrstevnicou je súčasťou objektu, všetko nad je okolie objektu.

Okrem toho, že tento prístup eliminuje nedostatky explicitných kontúr, prináša aj množstvo výhod. Rozšírenie do tretej dimenzie nie je problémom. V prípade segmentovania v 3D je pohyb reprezentovaný pohybom hyperplochy pomocou rovnakej rovnice, ktorá sa používa pri pohybe kontúry.



Obr. 8: Ukážka pohybu implicitnej kontúry. *(Zdroj: [7])*

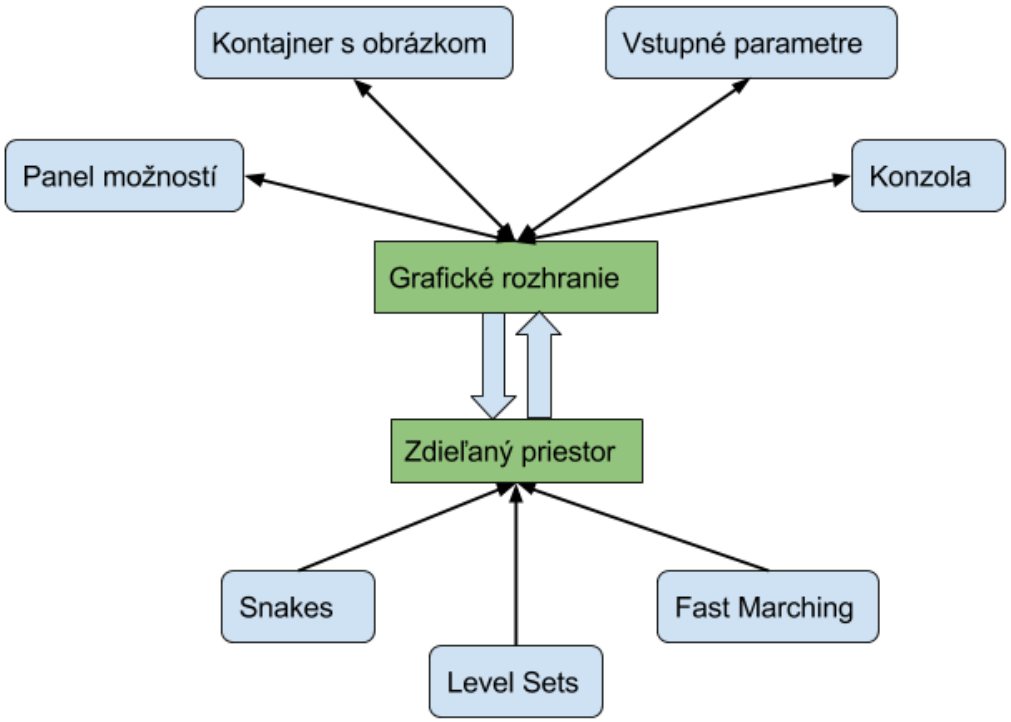
Pri zmene topológie nie je potrebná žiadna réžia. Pohybujúce sa prostredie zmení svoju topológiu, vzniká alebo zaniká úplne prirodzene. Spolu so zmenou implicitnej funkcie sa mení aj nultá vrstevnica.

Implicitný prístup oproti explicitnému rieši aj otázku príslušnosti, a to či daný bod patrí objektu alebo nie. Príslušnosť nám udáva znamienko funkčnej hodnoty.

Veľkou nevýhodou implicitných aktívnych kontúr je výpočtová náročnosť. Vzdialenosti jednotlivých bodov je potrebné prepočítavať v každom kroku, lebo vrstevnica medzičasom zmenila svoj tvar. Práve tento prepočet je jednou z hlavných príčin výpočtovej náročnosti.

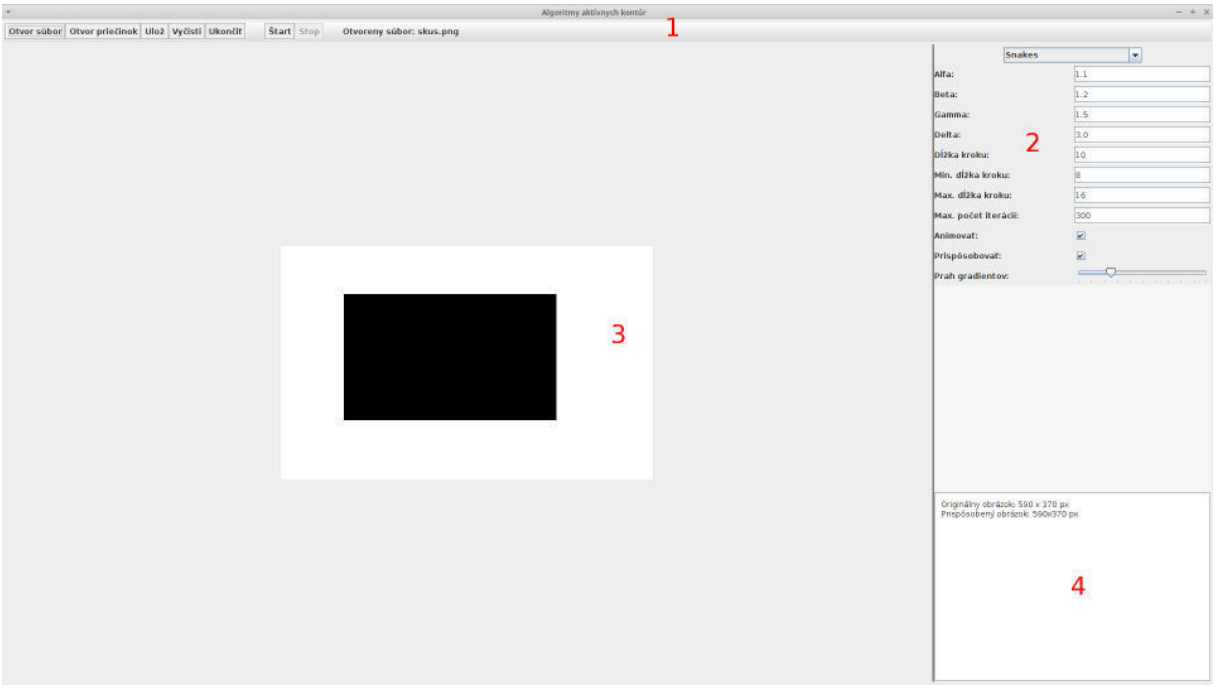
4 Princípy implementácie

Implementáciu algoritmov sme sa rozhodli naprogramovať v programovacom jazyku Java, ktorý je objektovo orientovaný. Program má jednoduchú štruktúru, ktorá je zložená z dvoch základných častí. Návrh architektúry môžeme vidieť na obrázku 9.



Obr. 9: Architektúra implentácie. *(Zdroj: Autor)*

Prvá časť je *grafické rozhranie*, ktoré má na starosti úlohy spojené so zobrazovaním údajov používateľovi. Grafické rozhranie umožňuje používateľovi: načítať obrázok, nastaviť vstupné parametre, spustiť respektíve ukončiť priebeh algoritmu a v neposlednom rade zobrazuje priebeh a výsledok segmentácie.



Obr. 10: Ukážka pohybu implicitnej kontúry. *(Zdroj: Autor)*

Ako môžeme vidieť na obrázku 10, grafické rozhranie sa skladá zo štyroch hlavných častí:

1. **Panel možností** - obsahuje tlačidlá pre načítanie obrázku alebo priečinku, spustenie a ukončenie výpočtu algoritmu a podobne.

2. **Kontajner s obrázkom** - zobrazuje načítaný obrázok a po spustení algoritmu vykresľuje priebeh a výsledok segmentácie.

3. **Vstupné parametre** - ktoré môže používateľ ľubovoľne meniť, a tým ovplyvniť výsledok segmentácie.

4. **Konzola** - má za účel informovať používateľa o tom, čo sa práve deje, respektíve ho upozorniť, že niečo nie je v poriadku.

Jednotlivé komponenty spolu dokážu komunikovať prostedníctvom modulu, z ktorého vychádzajú, teda z *grafického rozhrania*.

Druhou častou je *zdieľaný priestor*. Jeho úlohou je komunikácia s grafickým rozhraním, z ktorého čerpá všetky potrebné vstupné údaje, ktoré sú následne použité jednotlivými algoritmami. Okrem získania vstupných údajov je úlohou zdieľaného priestoru aj vy-

konať výpočty potrebné pred spustením samotných algoritmov. V tejto časti sa farebný obrázok mení na čiernobiely, vypočítavajú sa veľkosti gradientov alebo Chamferových vzdialeností a podobne. Zdieľaný priestor dostal svoj názov a účel hlavne kvôli tomu, že v tomto mieste sa stretávajú prvky potrebné pre všetky algoritmy.

Vylepšením, ktoré naša implementácia ponúka, je segmentácia množstva obrázkov naraz. V grafickom rozhraní sa totiž nachádza tlačidlo *„Načítaj priečinok"*, po kliknutí, ktorého nám dialógové okno dovolí vybrať ľubovoľný priečinok. Po vykonaní tejto akcie si program načíta všetky obrázky z priečinku, ktoré je možné segmentovať. Práve vďaka tomuto vylepšeniu sme mali možnosť testovať jednotlivé algoritmy. Pokiaľ by sme testovali všetky obrázky postupne na jednotlivých algoritmoch, je veľmi malá pravdepodobnosť, že by sme počiatočný bod, respektíve kontúru, zvolili v rovnakých miestach. Pokiaľ vyberieme segmentáciu priečinka a zvolíme si počiatočný bod, tento bod je taktiež počiatočným bodom pre všetky obrázky v priečinku. Ak sa používateľ pomýli a bod nezadá, implementácia automaticky vyberie vopred definované body. Pri algoritme Snakes je to kontúra vzdialená 10px od každej strany. V prípade Level Sets a Fast Marching algoritmu je to štvorica bodov vzdialená 5px od každého roku obrázka.

V prípade Snakes algoritmu je výstupom segmentácie kontúra, respektíve objekt v prípade Level Sets alebo Fast Marching algoritmu. Tento výsledok si môžeme uložiť, pričom pôvodný názov obrázku dostane príponu *„_vysledok"*. Ak používateľ zvolí segmentáciu priečinku, v danom priečinku sa vytvorí podpriečinok, ktorý bude obsahovať výsledky a vo zvolenom priečinku vznikne súbor obsahujúci časy výpočtu jednotlivých obrázkov spolu s ich názvom. Aby toho nebolo málo, tento súbor po zapísaní posledného výsledku zapíše na koniec zoznamu aj priemerný čas výpočtu. Implementácia vždy dopisuje na koniec súboru, takže pokiaľ používateľ spustí segmentáciu viackrát po sebe, uvidí všetky výpočty. Výhodou je fakt, že pokiaľ používateľ chce porovnať jednotlivé časy, stačí spustiť buď jeden algoritmus viackrát alebo viacero algoritmov raz. Jednotlivé zápisy v súbore sú oddelené názvom algoritmu a taktiež niekoľkými riadkami.

V nasledujúcej kapitole si vysvetlíme, ako sú jednotlivé algoritmy implementované a ako prebieha ich výpočet. Taktiež nezabudneme spomenúť výhody a nevýhody jednotlivých algoritmov.

4.1 Použité operátory

V tejto podkapitole nájdeme vysvetlenie ku každému potrebnému operátoru, ktorý je použitý v našej implementácii. Jednotlivé algoritmy nepotrebujú použiť všetky operátory, ale len niektoré z nich. Tieto operácie je potrebné vykonať ešte pred samotným spustením algoritmu, lebo upravujú vstupné údaje tak, aby s nimi mohli jednotlivé algoritmy bez problémov pracovať. Poďme si teda jednotlivé operátory popísať.

Prevod farebného obrázka na čiernobiely

Aby sme previedli farebný obrázok na čiernobiely, musíme sčítať hodnoty jednotlivých farieb a predeliť veľkosťou farebnej škály. Ako bolo spomenuté v úvode, obrázky sa skladajú z troch farieb a to červenej, zelenej a modrej, pričom každá z farieb nadobúda hodnotu od 0 po 255. Tieto hodnoty sčítame a predelíme troma.

$$šedá = \frac{červená + zelená + modrá}{3} \quad (1)$$

Sobelov operátor

Sobel operátor meria priestorový gradient v danom bode obrazu a dôraz kladie na oblasti s veľkou frekvenciou, ktoré zodpovedajú hranám. Používa sa na nájdenie veľkosti gradientu pre daný bod v obraze a my ho použijeme v implementácii Snakes algoritmu. Sobel operátor pracuje len s čiernobielym obrazom. Pokiaľ táto podmienka nie je splnená, je potrebné previesť obraz na čiernobiely, napríklad použitím metódy spomenutej vyššie.

Sobel operátor má tvar matice o veľkosti 3x3, ktoré reprezentujú os x a y. Výpočet gradientu prebieha v dvoch krokoch. Najprv sa bod a jeho okolie prepočíta po osi x, následne po osi y. Jednotlivé matice môžeme vidieť v tabuľke 1.

Tabuľka 1: Sobel operátor na osi x a osi y. (*Zdroj: [8]*)

-1	0	+1
-2	0	+2
-1	0	+1

Os x

+1	+2	+1
0	0	0
-1	-2	-1

Os y

Sčítaním prenásobenia týchto tabuliek získame výslednú tabuľku rovnakej veľkosti, ktorú aproximujeme podľa vzorca 2 [8].

Tabuľka 2: Výpočet gradientu pomocou Sobel operátora. (*Zdroj: [8])*

P_1	P_2	P_3
P_4	P_5	P_6
P_7	P_8	P_9

$$\begin{aligned} |G| &= |(P_1 + 2 * P_2 + P_3) - (P_7 + 2 * P_8 + P_9)| + \\ &\quad |(P_3 + 2 * P_6 + P_9) - (P_1 + 2 * P_4 + P_7)| \end{aligned} \tag{2}$$

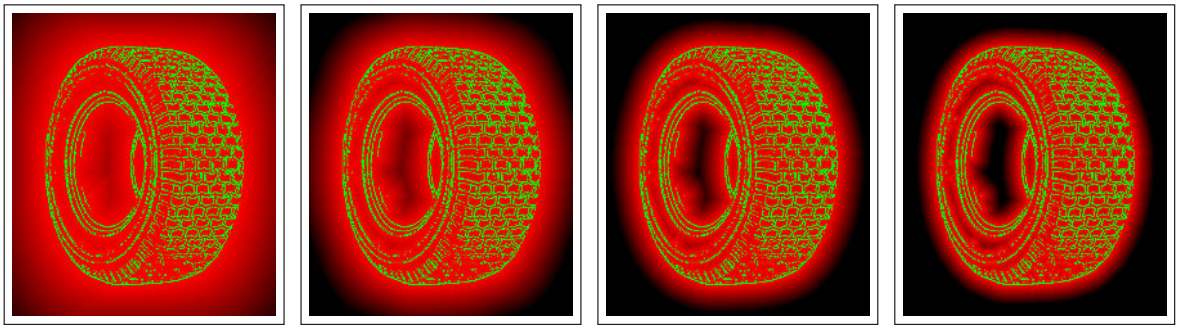
Chamferová vzdialenosť

Metódu Chamferovej vzdialenosti sme použili pri implementácií Snakes algoritmu na výpočet vzdialenosti daného bodu k najbližšiemu binárnemu gradientu. Táto metóda používa na prepočet vzdialenosti takzvané masky, ktorými daný bod spolu s jeho okolím prenásobuje. Výpočet vzdialenosti prebieha v dvoch smeroch, a to zhora nadol a naopak, zdola nahor. Masky sú symetrické, preto stačí zadať len polovicu masky, ktorá sa následne prevráti. Získame tak aj druhú stranu a vznikne mapa. V tabuľke 3 sú ukážky Chamferových másk, ktoré sme implementovali. Pri výpočte je však použitá len jedna maska, konkrétne maska o veľkosti *3x3*.

Na obrázku 11 sú zobrazené výsledky jednotlivých másk. Zelenou farbou sú označené gradienty, respektíve hrany objektu. Červenou farbou je označená vzdialenosť bodu od najbližšej hrany. Čím je intenzita farby vačšia, tým sa bod nachádza bližšie ku gradientu. [9]

Tabuľka 3: Tabuľka masiek Chamferových vzdialeností rôzných veľkostí. (*Zdroj: [9]*)

1 1 1 1 u - - - - *Chess (zhora nadol)*	- - - - u 1 1 1 1 *Chess (zdola nahor)*
4 3 4 3 u - - - - *3x3 (zhora nadol)*	- - - - u 3 4 3 4 *3x3 (zdola nahor)*
- 11 - 11 - 11 7 5 7 11 - 5 u - - - - - - - - - - - - *5x5 (zhora nadol)*	- - - - - - - - - - - - u 5 - 11 7 5 7 11 - 11 1 11 - *5x5 (zdola nahor)*
- 43 38 - 38 43 - 43 - 27 - 27 - 43 38 27 17 12 17 27 38 - - 12 u - *7x7 (zhora nadol)*	- u 12 - - 38 27 17 12 17 27 38 43 - 27 - 27 - 43 - 43 38 - 38 43 - *7x7 (zdola nahor)*



(a) Chess (b) 3x3 (c) 5x5 (d) 7x7

Obr. 11: Ukážka výsledkov prepočítania Chamferových vzdialeností. *(Zdroj: Autor)*

Výpočet gradientov

Aj vyššie spomenutý operátor Sobel slúži na výpočet gradientov, tentokrát však použijeme odlišnú metódu. Sobel operátor sme použili pri implementácii Snakes algoritmu a tento spôsob použijeme pri implementácii Level Sets a Fast Marching algoritmov. Hodnotu gradientu pre bod P získame pomocou vzorca 3.

Tabuľka 4: Výpočet gradientu. *(Zdroj: Autor)*

	y_1	
x_1	G	x_2
	y_2	

$$G = \sqrt{x^2 + y^2} \tag{3}$$

pričom $x = (x_1 - x_2)/2$ a $y = (y_1 - y_2)/2$.

5 Snakes algoritmus

Kapitola obsahuje všetky potrebné informácie ohľadom algoritmu Snakes. Avšak v tejto časti nenájdeme žiadne vysvetlenia princípov algoritmu, len si popíšeme základ formou matematického vyjadrenia. Princípy algoritmu si povieme v podkapitole 5.2, ktorá je zameraná na implementáciu tohto algoritmu. Poďme si teda algoritmus predstaviť.

Snakes algoritmus bol prvý krát popísaný v roku 1988 trojicou matematikov: *Michael Kass*, *Andrew Witkin* a *Demetri Terzopoulos*. Tento algoritmus patrí do kategórie explicitných aktívnych kontúr a ako už bolo spomenuté, algoritmus pracuje na princípe minimalizácie vnútorných a vonkajších energií. Snakes algoritmus reprezentuje kontúru parametricky ako $v(s) = (x(s), y(s))$, ktorej parametrom je bod kontúry s. Matematické vyjadrenie algoritmu má tvar:

$$E^*_{snake} = \int_0^1 E_{int}(v(s)) + E_{image}(v(s)) + E_{con}(v(s))ds \tag{4}$$

kde E_{int} reprezentuje vnútornú energiu, E_{image} je energiou obrázku (vonkajšia) a E_{con} reprezentuje energie obmedzení. Na čo slúžia vnútorné a vonkajšie energie sme si už povedali, preto si len rozpíšeme jednotlivé energie vzorcom a ako fungujú v praxi si vysvetlíme v praktickej časti. Vnútorná energia má tvar:

$$E_{int} = \frac{(\alpha(s)|v_s(s)|^2 + \beta(s)|v_{ss}(s)|^2))}{2} \tag{5}$$

a je zložená z dvoch častí $\alpha(s)$ a $\beta(s)$, pričom v_s a v_{ss} je prvá a druhá derivácia bodu $v(s)$. Obe časti ovplyvňujú pohyb kontúry, ale pokiaľ prvá časť $\alpha(s)$ núti kontúru, aby sa pohybovala ako membrána, druhá ju núti k pohybu ako doska. Inými slovami, tieto sily ovplyvňujú tvar kontúry konkrétne jej hladkosť a pružnosť. Napríklad nie vždy je požadované, aby kontúra okolo objektu vytvárala ostré hrany, ale naopak, očakávame jemný prechod. Práve $\beta(s)$ ovplyvňuje tvar kontúry v ostrých zmenách hrany objektu a pokiaľ nastavíme túto hodnotu na 0, aj samotná kontúra nadobudne ostré hrany.

Vonkajšiu energiu vieme opísať ako:

$$E_{image} = \omega_{line} * E_{line} + \omega_{edge} * E_{edge} + \omega_{term} * E_{term} \tag{6}$$

kde zložky ω sú váhy jednotlivých parametrov. Parameter E_{line} (vzorec 7) ovplyvňuje prilnavosť kontúry ku hrane objektu. V závislosti od znamienka váhy ω je kontúra priťahovaná buď k svetlým alebo k tmavým hranám obrazu

$$E_{line} = I(x, y) \tag{7}$$

Parameter E_{edge} (vzorec 8) ovplyvňuje, k akým veľkým gradientom sa má kontúra prilepiť. Čím je hodnota väčšia, tým je potrebná väčšia hodnota gradientu pre prilepenie kontúry.

$$E_{edge} = -| \bigtriangledown I(x, y)|^2 \tag{8}$$

Úlohou ďalšieho parametru E_{term} (vzorec 9) je detekovať konce hrán, poprípade ostrých rohov. Takáto detekcia prebieha pri mierne rozmazanom obraze. Majme $C(x, y) = G_\sigma(x, y) * I(x, y)$, čo predstavuje rozmazaný obraz a $\theta = tan^{-1}(C_y/C_x)$ je uhol gradientu. Ďalej majme $\mathbf{n} = (cos\theta, sin\theta)$ a $\mathbf{n}_\perp = (-sin\theta, cos\theta)$ sú vektory pozdĺž a kolmo na smer gradientu. Úroveň zakrivenia kontúry v bode C(x,y) môžeme zapísať ako:

$$\begin{aligned} E_{term} &= \frac{\partial\theta}{\partial\mathbf{n}_\perp} \\ &= \frac{\partial^2 C/\partial\mathbf{n}_\perp^2}{\partial C/\partial\mathbf{n}} \\ &= \frac{C_{yy}C_x^2 - 2C_{xy}C_xC_y + C_{xx}C_y^2}{(C_x^2 + C_y^2)^{3/2}} \end{aligned} \tag{9}$$

Viac o tomto algoritme sa môžeme dočítať v práci, kde bol tento algoritmus prvý

krát opísaný [10].

5.1 Parametre algoritmu Snakes

Pred samotným spustením algoritmu je potrebné nastaviť určité parametre. Grafické rozhranie nám ponúka vopred definované hodnoty, ale tieto hodnoty môžeme ľubovoľne nastavovať a tým ovplyvňovať výsledky segmentácie. Nasledujúci zoznam obsahuje všetky parametre, ktoré program obsahuje a ich vplyv na výsledok.

- **Alfa** - reprezentuje jednotnosť, teda udržuje kontúru pokope. Zvýšením tejto hodnoty zabraníme, aby sa kontúra dotýkala rovnakého miesta viackrát.
- **Beta** - reprezentuje zakrivenie kontúry. Zvýšením tejto hodnoty ovplyvníme zakrivenie kontúry, ktorá si následne udrží hladký tvar. Znížením hodnoty naopak kontúra nadobudne hranatý tvar.
- **Gamma** - priťahuje kontúru ku gradientom. Zvýšením tejto hodnoty zabezpečíme väčšiu príťažlivosť kontúry ku gradientom.
- **Delta** - udržuje kontúru pri gradientoch.
- **Dĺžka kroku** - je ideálna vzdialenosť bodov v kontúre.
- **Minimálna dĺžka kroku** - je vzdialenosť bodov, po prekročení ktorej body v kontúre zaniknú.
- **Maximálna dĺžka kroku** - je vzdialenosť bodov, po prekročení ktorej body v kontúre vzniknú. Táto vzdialenosť sa taktiež nachádza medzi jednotlivými bodmi vo výslednej kontúre.
- **Maximálny počet iterácií** - chráni algoritmus pred zacyklením. Po prekročení tejto hodnoty sa program zastaví a kontúra je považovaná za výsledok. Pokiaľ algoritmus nenájde včas výslednú kontúru objektu, zvýšením tejto hodnoty pridáme počet iterácií, a tým umožníme algoritmu výsledok nájsť.
- **Animovať** - priebeh algoritmu. Pokiaľ je táto možnosť zaškrtnutá, používateľ môže vidieť, ako sa kontúra postupne vyvíja, v opačnom prípade sa rozbrazí len výsledok.

- **Prispôsobiť** - kontúru, respektíve ak je táto možnosť zaškrtnutá, tak jednotlivé body v kontúre môžu vznikať a zanikať na základe vzdialeností udávaných vyššie.
- **Prah gradientov** - je prahová hodnota gradientov, ktorá sa použije pri výpočte binárnych gradientov. Hodnotu je potrebné zvýšiť, ako kontúra nedosiahla hranu objektu a znížiť, ak túto hranu prekročila.

5.2 Implementácia algoritmu Snakes

Ešte pred samotným spustením algoritmu je potrebné vykonať niekoľko výpočtov. V implementácii si vytvoríme niekoľko matíc (dvojrozmerných polí). Prvým krokom je previesť farebný obrázok na maticu čiernobielych hodnôt, pričom si do premennej uložíme hodnotu najväčšieho gradientu.

V nasledujúcom kroku si vytvoríme maticu binárnych gradientov. Binárny gradient predstavuje informáciu, či sa v danom bode nachádza gradient alebo nie. Na získanie týchto informácií potrebujeme hodnotu najväčšieho gradientu, ktorú sme získali predošlým výpočtom a hodnotu *prahu gradientov*, ktorá je získaná z grafického rozhrania. Hodnotu binárneho gradientu teda získame ako

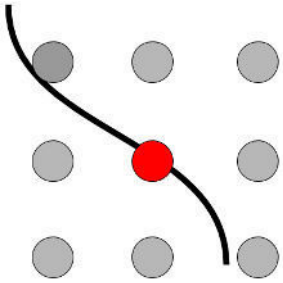
$$\textit{binárny gradient} = \frac{\textit{prahová hodnota} * \textit{najväčší gradient}}{100} \tag{10}$$

V ďalšom kroku sa vytvorí matica vzdialenosti daného bodu k najbližšiemu binárnemu gradientu. Na výpočet tejto vzdialeností sme použili metódu Chamferovej vzdialenosti.

Vstupom do samotného algoritmu sú vstupné parametre získané z grafického rozhrania, matica čiernobielych hodnôt, matica vzdialeností, matica binárnych gradientov a kontúra. Kontúra je reprezentovaná ako sústava bodov a zadáva ju používateľ okolo objektu, ktorý má byť segmentovaný. Čím bližšie sa kontúra nachádza, tým je čas výpočtu kratší.

Hlavný výpočet prebieha v cykle, ktorý sa vykonáva pokiaľ segmentácia nájde hranu objektu, alebo nie je prekročený maximálny počet iterácií. V každej iterácii sa nachá-

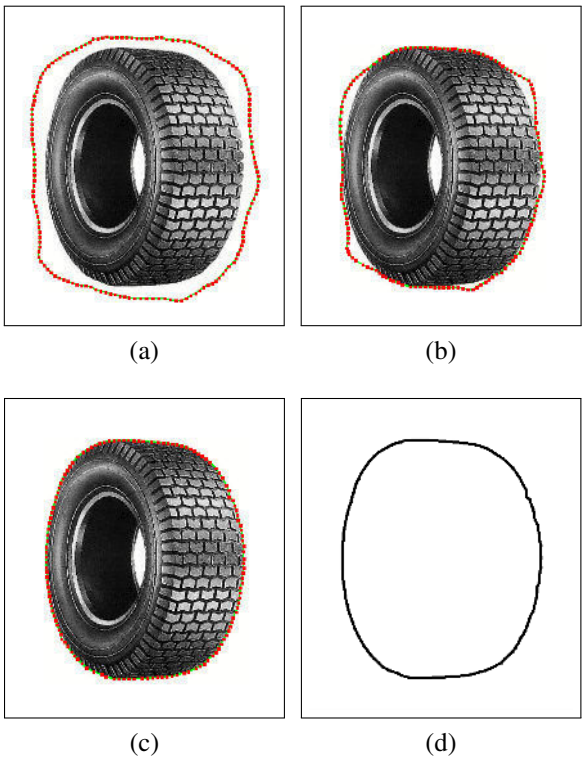
dza ďalší cyklus, ktorý sa stará o prepočet kontúry. Tento cyklus totiž prechádza každým bodom kontúry a kontroluje, či je možné daný bod posunúť. V implementácii máme boolovskú premennú, ktorá zmení svoju hodnotu na *pravda*, ak aspoň jeden bod z kontúry zmení svoju pozíciu. V opačnom prípade, ak na konci iterácie celou kontúrou ani jeden bod nezmenil pozíciu, kontúra je považovaná za výsledok.



Obr. 12: Bod kontúry (červený bod) spolu s okolitými bodmi (šedé body). *(Zdroj: Autor)*

Ktorým smerom sa kontúra pohne závisí od prepočítania daného bodu kontúry a jeho okolia. Pokiaľ sa v okolí nájde bod s menšou energiou, kontúra sa presunie na pozíciu daného bodu. Tento postup môžeme vidieť na obrázku 12.

Priebeh výpočtu algoritmu môžeme vidieť na obrázku 13, kde obrázok 13(a) je pôvodný obrázok, 13(b, c) je priebeh algoritmu a 13(d) je výsledná kontúra. Okrem výslednej kontúry používateľ získa aj informáciu o čase, za ktorý bol algoritmus schopný získať výsledok.



Obr. 13: Priebeh segmentácie Snakes algoritmu. *(Zdroj: Autor)*

5.3 Výhody implementácie algoritmu Snakes

Veľkou výhodou je jednoduchosť algoritmu, ktorá sa odráža na jeho výkonnosti a má vplyv na celkový výpočtový čas. Čas trvania výpočtu je neporovnateľne rýchlejší s ostatnými implementovanými algoritmami.

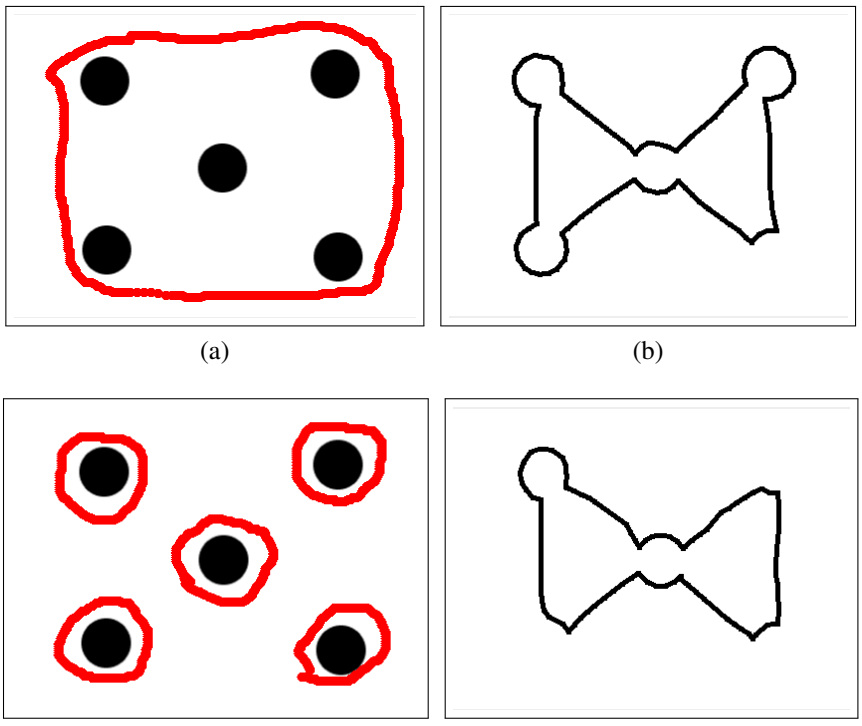
5.4 Nevýhody implementácie algoritmu Snakes

To, čo algoritmus vynahrádza jednoduchosťou, stráca na kvalite výsledkov. Pokiaľ sa segmentujú náročnejšie objekty, respektíve kvalita obrazu nie je dostatočná, výsledky sú mnohokrát nežiadúce. Taktiež algoritmus má veľký problém segmentovať objekty, ktoré sú umiestené na kraji obrázku, ako to je vidieť na obrázku 14.

(a) (b)

Obr. 14: Ukážka segmentácie objektu na kraji pomocou Snakes algoritmu. *(Zdroj: Autor)*

Algoritmus tiež nedokáže segmentovať viacero objektov naraz. Túto segmentáciu môžeme vidieť na obrázku 15.



(a) (b)

(c) (d)

Obr. 15: Ukážka segmentácie viacerých objektov pomocou Snakes algoritmu. *(Zdroj: Autor)*

Ďalšou nevýhodou je test príslušnosti. Inými slovami, nie je možné povedať, či sa daný bod nachádza vo vnútri objektu alebo v jeho okolí. Algoritmus nám ponúka len informácie o hrane objektu.

6 Level Sets algoritmus

Tak ako pri predošlom algoritme, aj v tomto prípade sa najskôr oboznámime s teóriou a matematickým vyjadrením. Postupne prejdeme k implementácii, kde sa dozvieme ako algoritmus pracuje, jeho výhody a nevýhody. Poďme si teda predstaviť Level Sets algoritmus.

Algoritmus vznikol v roku 1988 a za jeho tvorcov považujeme *Stanleyho Oshera* a *Jamesa Sethiana* a patrí do kategórie implicitných aktívnych kontúr. Algoritmus nereprezentuje kontúru ako parametrickú krivku, ale ako nultú vrstevnicu, čo môžeme vidieť na obrázku 16.

Predpokladajme, že máme počiatočnú pozíciu rozhrania, ako nultú vrstevnicu funkcie vyššej dimenzie ϕ. Vieme s určitosťou povedať, kde sa dané rozhranie nachádza a taktiež predpovedať jeho pohyb. Treba podotknúť, že nultá vrstevnica sa vždy zhoduje s pohybujúcim sa rozhraním a to znamená že:

$$\phi(x(t), t) = 0. \tag{11}$$

Pričom $x(t)$ je následný pohyb bodu rozvíjajúceho sa rozhrania a $t\epsilon[0, \infty)$ Derivovaním zloženej funkcie získame:

$$\phi_t + \nabla\phi(x(t), t) \cdot x'(t) = 0. \tag{12}$$

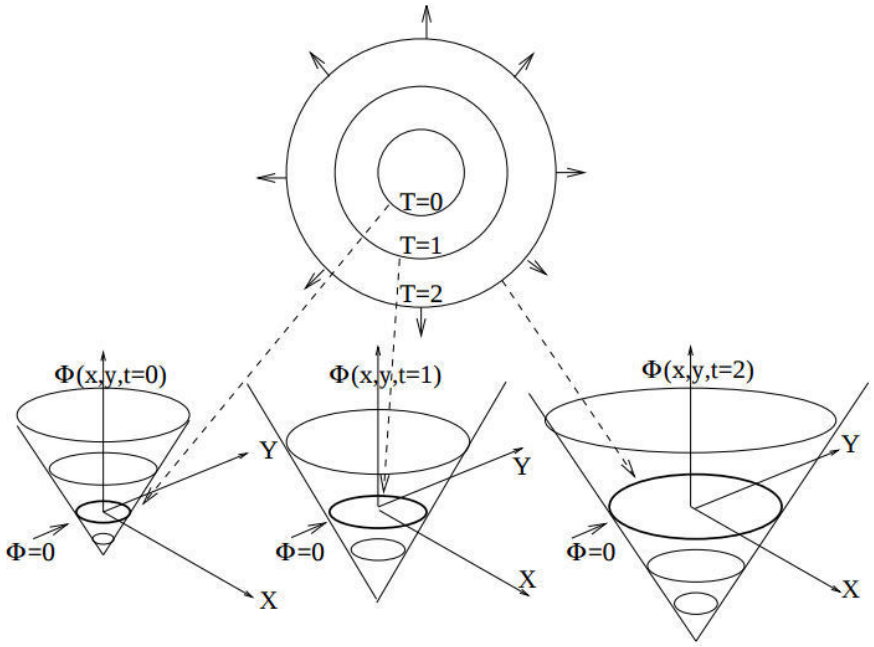
Majme teda F, ktoré reprezentuje rýchlosť v smere normály, potom $x'(t) \cdot n = F$, kde $n = \phi_t/|\phi_t|$. Takto získame funkciu vývoja ϕ, ktorá má tvar:

$$\phi_t + F|\nabla\phi| = 0, \tag{13}$$

$$\phi(x, t = 0). \tag{14}$$

Veľkou výhodou tohto prístupu je fakt, že rozhranie môže ľubovoľne meniť smer

pohybu rozhrania a to zmenou znamienka rýchlosti F. Pokiaľ $F > 0$ rozhranie sa pohybuje dopredu, v opačnom prípade ak $F < 0$, rozhranie sa pohybuje dozadu.



Obr. 16: Ukážka segmentácie viacerých objektov pomocou Snakes algoritmu. *(Zdroj: [13])*

Toto je len veľmi stručný prehľad takzvaného problému počiatočnej hodnoty. Podrobnejšie vysvetlenie teórie môžeme nájsť v [11, 12, 13, 14].

6.1 Parametre algoritmu Level Sets

Teraz si povieme, ako naša implementácia algoritmu Level Sets funguje. Avšak skôr než začneme, podme si popísať vstupné parametre potrebné pre správne fungovanie algoritmu.

- **Advekcia** - je v podstate rýchlosť vývoja kontúry. Zvyšovaním tejto hodnoty sa môže zvýšiť rýchlosť segmentácie, avšak treba byť opatrný. Pokiaľ je hodnota príliš vysoká, môže sa stať, že segmentácia bude prirýchla a minie kontúru.
- **Zakrivenie** - predstavuje váhu zakrivenia vyvíjajúcej sa kontúry. Zvýšením tejto hodnoty sa berie väčší ohľad na tvar kontúry, ktorá si udrží oblé tvary v hranách objektu. Znížením tejto hodnoty kontúra nadobudne hranatý tvar.

- **Tolerancia šedej farby** - porovnáva šedú hodnotu aktuálneho bodu s bodom ďalšieho kroku vývoja kontúry. Pokiaľ v ďalšom bode je táto hodnota prekročená, zavedie sa do tohto bodu takzvaná pokuta a segmentácia sa do tohto bodu už nikdy nevráti.
- **Konvergencia** - táto hodnota slúži ako kontrola chyby. Pokiaľ sa hodnota rozdielu medzi dvoma iteráciami dostane pod túto úroveň, algoritmus sa zastaví. Túto hodnotu je potrebné zvýšiť, pokiaľ sa kontúra nezastaví na hrane objektu alebo znížit, pokiaľ sa kontúra zastaví priskoro.

6.2 Implementácia algoritmu Level Sets

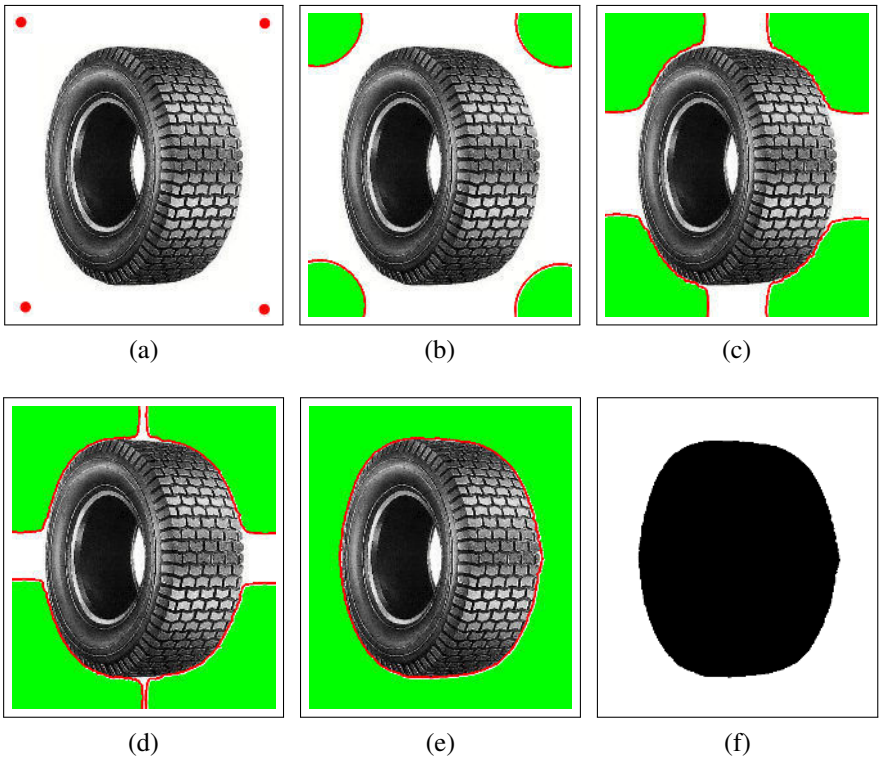
Aj v tomto prípade je potrebné urobiť pár výpočtov ešte pred samotným spustením algoritmu. Tieto výpočty nám pomôžu získať maticu čiernobielych hodnôt z pôvodného obrázku, maticu gradientov a priemernú šedú hodnotu. Priemernú hodnotu šedej získame z čiernobielych hodnôt počiatočných bodov. V prípade, že je vstupný bod len jeden, priemerná hodnota šedej je práve šedou hodnotou daného bodu.

Keď už máme všetky potrebné údaje, môžeme spustiť algoritmus. Ten si na začiatku inicializuje potrebné premenné a rozdelí obraz na 5 vrstiev, ktoré sú reprezentované ako zoznamy bodov:

- vzdialené body mimo objektu,
- body v blízkosti hrany objektu,
- body hrany objektu,
- body objektu v blízkosti hrany,
- body vo vnútri objektu.

Princíp implementácie je podobný ako u algoritmu hada. Všetky výpočty prebiehajú v hlavnom cykle, ktorý pracuje s vyššie spomenutými vrstvami. V prvom kroku sa do zoznamu bodov objektu priradia naše vstupné body a ich susedia sa priradia k hrane objektu. Ako prebieha výpočet, medzi body objektu sa pridávajú ďalšie body a objekt sa tak rozrastá. Predstavme si to ako gumičku, ktorú položíme niekde do obrazu a začne sa rozťahovať.

Gumička drží objekt pohromade a zabraňuje tak „pretekaniu"v prípade, že v hrane objektu sa nachádzajú diery. Vzhľadom na fakt, že výpočet prebieha iteratívne, je v každej iterácii potrebné prepočítať všetky body v blízkosti hrany objektu. Takto vieme povedať, kam sa hrana posunie v nasledujúcej iterácií. Každou iteráciou sa prepočíta hodnota zmeny hrany a porovná sa s predošlou. Ak je zmena menšia ako požadovaná presnosť, algoritmus končí.



Obr. 17: Priebeh segmentácie Level Sets algoritmu. *(Zdroj: Autor)*

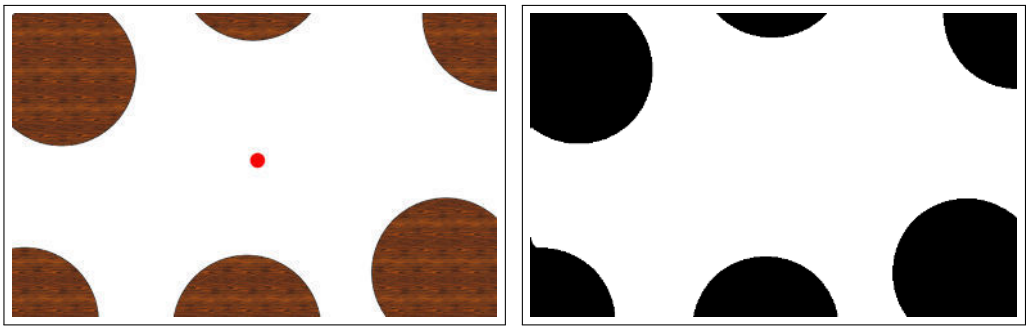
Na obrázku 17 môžeme vidieť priebeh výpočtu algoritmu, pričom (a) obsahuje nami zvolené počiatočné body. V nasledujúcom obrázku už je vidieť, ako sa objekt pomaly rozrastá (zelená plocha) a tlačí pred sebou hranu (červená kontúra). V tomto konkrétnom prípade segmentácia nájde len obrys pneumatiky, no pokiaľ by sme zvolili bod záujmu aj do jej stredu, tak by nám segmentácia vrátila celú plochu pneumatiky.

Implementácia má v sebe parametre, ktoré používateľ nedokáže ovplyvniť ako maximálny počet iterácií alebo frekvenciu vykresľovania. Avšak vzhľadom na to, že jeden z parametrov ovplyvňuje rýchlosť vývoja, počet iterácií nie je podstatný. Je to skôr len ako bezpečnostný prvok proti zacykleniu. Frekvenciu vykresľovania sme nastavili na každých 100 iterácií, pričom práve každá stá iterácia sa zobrazí používateľovi ako medzivýsledok. Algoritmus je možné stopnúť a spustiť odznova, no nie je možné ho pozastaviť a spustiť od

bodu, v ktorom prestal.

6.3 Výhody implementácie algoritmu Level Sets

Medzi výhody Level Sets algoritmu patrí kvalita výsledkov. Schopnosť algoritmu meniť smer šírenia rozhrania dovoľuje prepočítavať už prepočítané hodnoty, a teda môže tým zvýšiť kvalitu výsledku. Oproti Snakes je tento algoritmus vhodný aj na segmentáciu zložitejších objektov, ktoré sa môžu nachádzať kdekoľvek v obraze a ich počet nie je rozhodujúci. Túto výhodu môžeme vidieť na obrázku 18, pričom 18 (a) obsahuje niekoľko objektov rozdelených po okraji obrázku a 18 (b) je výsledok segmentácie.



(a) (b)

Obr. 18: Ukážka segmentácie objektu na kraji pomocou Snakes algoritmu. *(Zdroj: Autor)*

Ďalšou výhodou oproti Snakes je test príslušnosti. Tento algoritmus poskytuje informácie, či sa daný bod nachádza v objekte, na jeho hrane alebo mimo objektu.

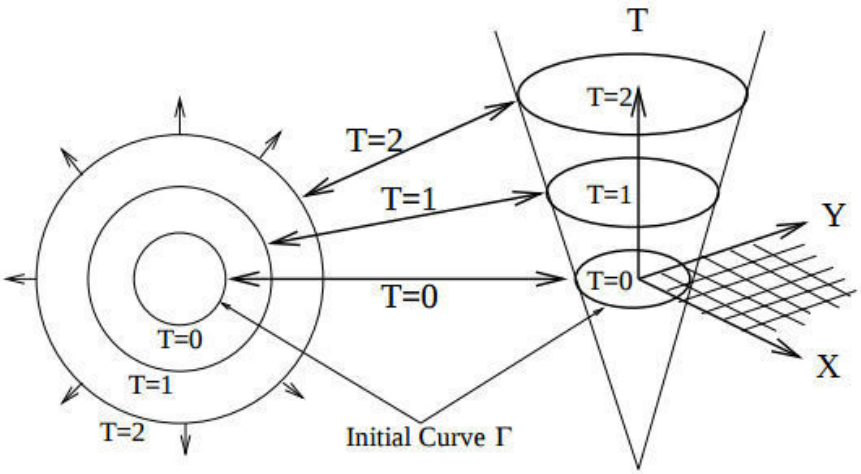
6.4 Nevýhody implementácie algoritmu Level Sets

Hlavnou a základnou nevýhodou je vypočtová a časová náročnosť. Oproti ostatným algoritmom je to najnáročnejší algoritmus na výpočet. Práve pre svoju výpočtovú náročnosť sa prišlo s novým prístupom, takzvaným Fast Marching, ktorý si opíšeme v ďalšej kapitole.

7 Fast Marching algoritmus

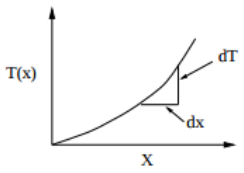
Ako sme zvyknutí už z predošlých algoritmov, aj tu si najprv vysvetlíme matematické vyjadrenie a postupne sa dostaneme k implementácii. Fast Marching bol prvýkrát popísaný v práci *A Fast Marching Level Set Method for Monotonically Advancing Fronts* v roku 1995. Rovnako ako Level Sets algoritmus, aj Fast Marching predstavuje krivku ako nultú vrstevnicu. Môžeme to vidieť na obrázku 19.

Predstavme si uzavretú krivku Γ v rovine, ktorá sa pohybuje rýchlosťou F. Predpokladajme, že rýchlosť $F > 0$, pretože sa hranica pohybuje smerom dopredu. Jedným zo spôsobov ako charakterizovať polohu hranice, je vypočítať čas príchodu $T(x, y)$, čo reprezentuje čas, za ktorý hranica dosiahne bod (x, y).



Obr. 19: Pohyb krivky. *(Zdroj: [13])*

Rovnicu na získanie času príchodu získame deriváciou jednoduchého vzorca na výpočet vzdialenosti, teda deriváciou *vzdialenosť = rýchlosť ∗ čas*.



Obr. 20: Derivácia vzorca na výpočet vzdialenosti. *(Zdroj: [13])*

získame

$$dx = F(dT) \tag{15}$$

a preto

$$1 = F\frac{dT}{dx} \tag{16}$$

Z $\frac{dT}{dx}$ sa stane gradient, a teda výsledný vzorec má hodnotu

$$|\nabla \phi|F \quad = \quad 1 \tag{17}$$

pričom $T = 0$ na krivke Γ. Oproti Level Sets, tento spôsob nedokáže meniť smer pohybu ľubovoľne počas priebehu, ale smer je pevne daný. Toto bolo krátke priblíženie takzvaného problému hodnoty hrany. Viac o tejto téme sa môžeme dočítať v [13, 14, 15].

7.1 Parametre algoritmu Fast Marching

- **Prahová hodnota šedej farby** - udáva maximálny rozdiel šedej hodnoty medzi aktuálnym bodom a počiatočným bodom. Ak je v danom bode rozdiel vačší, segmentácia pokračuje iným smerom. Túto hodnotu je potrebné zvýšiť, ak je obraz silne kontrastný a znížiť, ak je obraz naopak málo kontrastný.
- **Prahová hodnota vzdialenosti** - udáva, do akej vzdialenosti je povolené expandovať segmentáciu v jednej iterácii. Zvýšenie tejto hodnoty síce urýchli proces segmentácie, no na druhej strane znecitliví"segmentáciu. Pokiaľ je hodnota privysoká, môže nastať podobná situácia ako pri Level Sets algoritme a segmentácia minie kontúru.

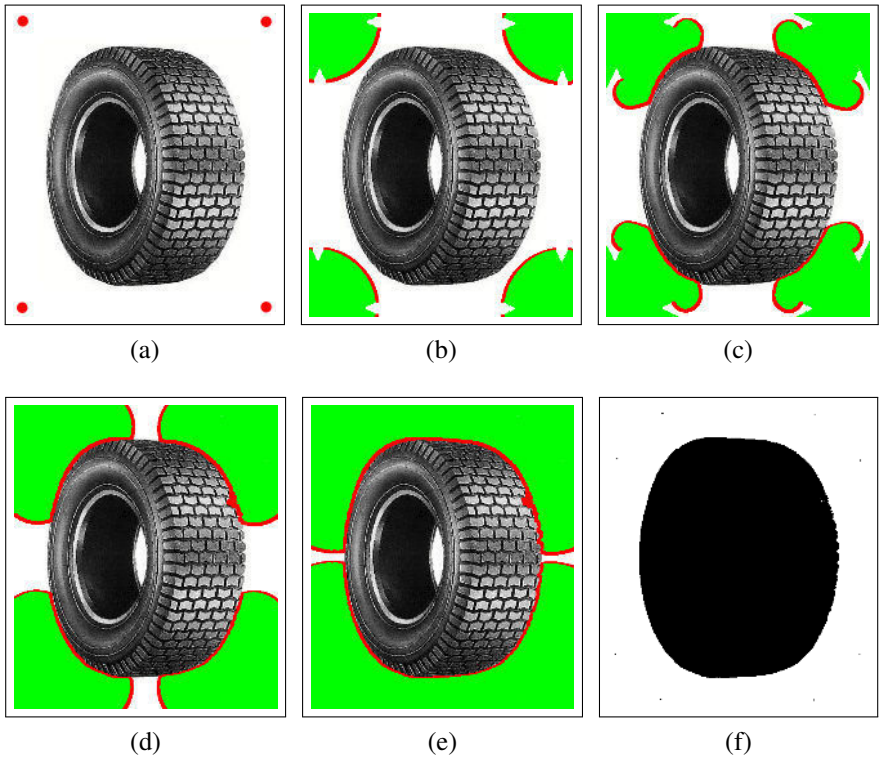
7.2 Implementácia algoritmu Fast Marching

Teraz si vysvetlíme princíp, ako funguje implementácia posledného algoritmu. Doteraz všetky implementácie sa držali určitého vzoru návrhu. Po načítaní obrázku prebehne pár výpočtov, ktorých výsledky spolu so vstupnými parametrami sú vstupom pre algoritmus. Ani táto implementácia sa od predošlých veľmi nelíši. Teraz tiež potrebujeme získať maticu čiernobielych hodnôt, maticu gradientov, priemernú hodnotu šedej a vstupné body.

Po spustení algoritmu sa opäť inicializujú potrebné premenné, tentokrát však obraz nie je rozdelený na vrstvy. Počas inicializácie sa vytvorí niekoľko matíc (dvojrozmerných

polí) o veľkosti vstupného obrázka a vyplnia sa predvolenými hodnotami. Jednotlivé matice reprezentujú hodnoty času príchodu a maximálnu vzdialenosť hľadania hrany od daného bodu. Inými slovami, ku každému obrazovému bodu si vytvoríme dodatočné údaje, za aký čas dorazí hrana k danému bodu a do akej maximálnej vzdialenosti budeme hľadať hranu.

Priebeh algoritmu môžeme vidieť na obrázku 21. Predstavme si to ako keď sa valí voda do údolia. Pri stálom prúdení sa údolie zaleje vodou za určitý čas. V našom prípade však čas nie je až taký podstatný, ide skôr o výšku, do ktorej sa má údolie zaliať. V prípade, že zalejeme údolie do požadovanej výšky, voda nám bude reprezentovať pozadie. Všetko nad hladinou vody je objekt a miesto, kde sa voda dotýka pevniny je hrana objektu. V implementácii je požadovaná výška zaliatia údolia reprezentovaná, ako maximálny rozdiel hodnoty šedej farby aktuálneho bodu a nášho vstupného bodu. Všetky body, ktoré sú menšie ako tento rozdiel, sú priradené pozadiu, všetko väčšie je priradené objektu a medzi nimi sa nachádza hrana objektu.



Obr. 21: Priebeh segmentácie Fast Marching algoritmu. *(Zdroj: Autor)*

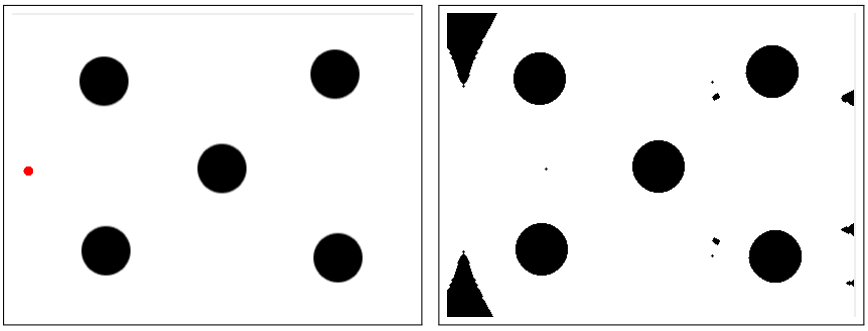
Tento algoritmus je citlivejší na detekciu hrán, avšak aj na pretekanie. Môže sa stať, že počas výpočtu sa kontúra dostane mimo objekt. Fast Marching algoritmus je aj podstatne rýchlejší a pri väčších obrázkoch svoj časový náskok zvyšuje.

7.3 Výhody implementácie algoritmu Fast Marching

Výhodou implementácie je rýchlosť. Algoritmus síce nie je taký rýchly ako *Snakes*, avšak oproti *Level Sets* algoritmu je omnoho rýchlejší. Pridaním ďalších počiatočných bodov zvýšime výpočtový čas, ten bude stále neporovnateľne menší ako pri *Level Sets* algoritme.

7.4 Nevýhody implementácie algoritmu Fast Marching

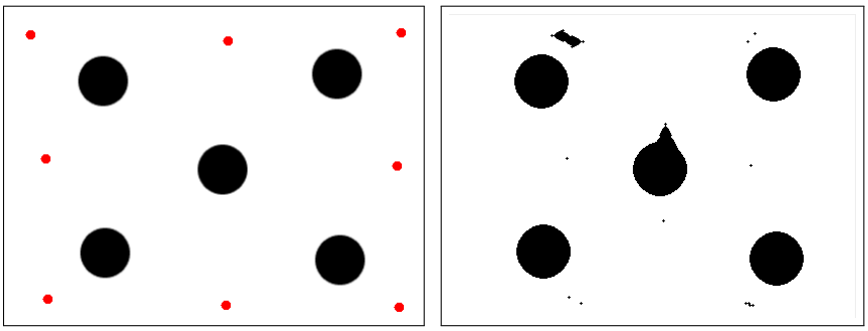
Základnou nevýhodou je možnosť chybného, respektíve neúplného výsledku, ako môžeme vidieť na obrázku 22 (b). To sa dá však ovplyvniť pridaním ďalších počiatočných bodov.



(a) (b)

Obr. 22: Segmentácia s jedným vstupným bodom. *(Zdroj: Autor)*

Avšak aj pridanie počiatočných bodov nie je zárukou kvalitného výsledku, ako môžeme vidieť na obrázku 23 (b). Preto z praktického hľadiska sa odporúča kombinácia Fast Marching a Level Sets algoritmov. Segmentácia teda začne Fast Marching algoritmom, ktorý je dostatočne rýchly a neúplný výsledok je vstupom pre Level Sets algoritmus, ktorý dokončí segmentáciu. Celkový čas priebehu algoritmu je niekde medzi samotnými priebehmi jednotlivých algoritmov. Zvyčajne sa výpočtový čas blíži skôr Fast Marching algoritmu.



(a) (b)

Obr. 23: Segmentácia s viacerými vstupnými bodmi. *(Zdroj: Autor)*

8 Segmentácie trasologických objektov

Teraz si ukážeme výsledky segmentácie na jednotlivých trasologických objektoch. Najprv si však povedzme niečo o spôsobe testovania. Ako vstupné údaje sme použili 30 obrázkov s trasologickými objektmi, ktoré sú rozdelené po 10 obrázkov do troch skupín a to na:

- **veľké** - šírka obrazu v rozsahu od 400px do 600px,
- **stredné** - šírka obrazu v rozsahu od 200px do 399px,
- **malé** - šírka obrazu v rozsahu od 100px do 199px.

Tieto rozlíšenia sme zvolili, pretože sú dostatočne veľké, aby na nich bolo vidieť priebeh segmentácie a taktiež obrázky nie sú zbytočne roztiahnuté pri obrazovkách s menším rozlíšením.

Na testovanie sme použili algoritmus hada, Level Sets, Fast Marching a kombináciu Fast Marching + Level Sets. Test bol spravodlivý voči všetkým implementáciam, takže žiadna z nich nebola zvýhodnená. Testované obrázky obsahovali výhradne trasologické objekty umiestnené v strede a na bielom pozadí. Úlohou testu bolo porovnať kvalitu výsledkov a výpočtový čas, za ktorý boli výsledky získane. Test sa skladal z troch krokov a to:

- segmentácia pomocou algoritmov explicitných kontúr so vstupnou kontúrou,
- segmentácia pomocou algoritmov implicitných kontúr s jedným vstupným bodom,
- segmentácia pomocou algoritmov implicitných kontúr so štyrmi vstupnými bodmi.

Keďže implementácia algoritmov explicitných kontúr sa líši od implementácií implicitných kontúr potrebnými vstupmi, rozhodli sme sa tieto segmentácie oddeliť. Taktiež nás zaujímalo, ako je segmentácia implicitnými kontúrami ovplyvnená počtom vstupných bodov. To je dôvod prečo sme rozdelili segmentáciu implicitnými kontúrami na dve kategórie.

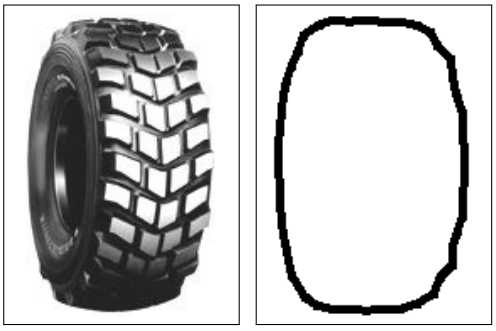
Vzhľadom na fakt, že sme celkovo testovali 10 obrázkov sedemkrát, z každej kategórie nám vzniklo 70 výsledkov. Kvôli veľkému počtu výsledkov je takmer nemožné ich

všetky ukázať v tejto práci. Rozhodli sme sa teda ukázať len jeden výsledok z každej kategórie. Účely jednotlivých segmentácií spolu s výsledkami a vyhodnotením si vysvetlíme v nasledujúcich podkapitolách. V závere kapitoly si popíšeme výsledky a zistenia jednotlivých testov.

8.1 Segmentovanie explicitnými kontúrami so vstupnou kontúrou

Úlohou tohto testu bolo otestovať segmentáciu trasologických objektov na desiatich obrázkoch z každej kategórie s použitím algoritmov explicitných aktívnych kontúr, konkrétne použitím Snakes algoritmu. Vstupom do algoritmu bola kontúra, ktorá sa nachádza presne 5px z každej strany od hrany obrázku.

Výsledné kontúry sú si podobné, preto uvedieme jeden príklad z každej kategórie, kde uvedieme veľkosť pôvodného obrázku a čas, za ktorý segmentácia dospela k výsledku.



(a) Originál (b) 75 ms

Obr. 24: Výsledok segmentácie malého obrázku použitím Snakes algoritmu. *(Zdroj: Autor)*

(a) Originál (b) 389 ms

Obr. 25: Výsledok segmentácie stredného obrázku použitím Snakes algoritmu. *(Zdroj: Autor)*

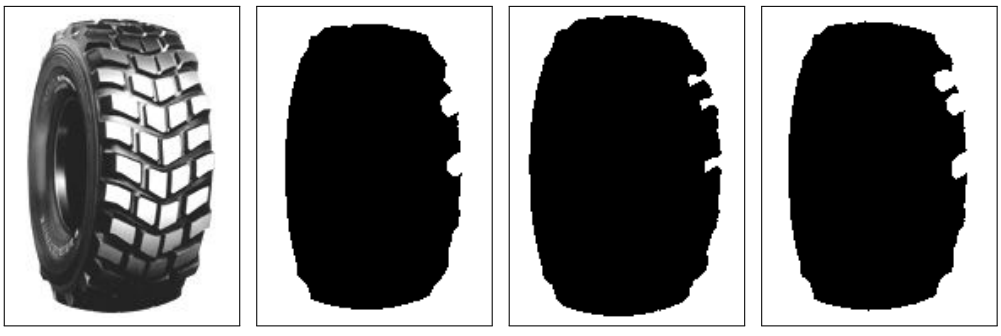
(a) Originál (b) 798 ms

Obr. 26: Výsledok segmentácie veľkého obrázku použitím Snakes algoritmu. *(Zdroj: Autor)*

8.2 Segmentovanie implicitnými kontúrami s jedným vstupným bodom

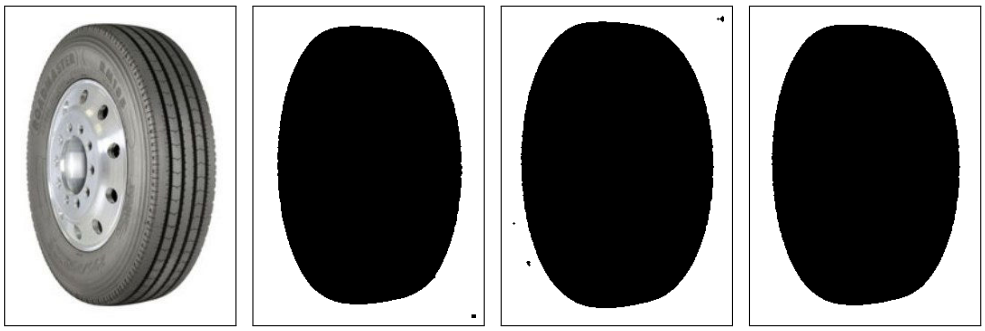
V tejto podkapitole nájdeme výsledky segmentácie použitím algoritmov implicitných aktívnych kontúr. Pri tejto segmentácií bol použitý jeden vstupný bod, umiestnený 5px od ľavého horného rohu obrázku. Každý obrázok obsahuje pôvodný obrázok a výsledky jednotlivých segmentácií spolu s časom výpočtu, pričom

- **(a)** - je pôvodný obrázok,
- **(b)** - výsledok segmentácie použitím Level Sets algortimu,
- **(c)** - výsledok segmentácie použitím Fast Marching algortimu,
- **(d)** - výsledok segmentácie použitím kombinácie Fast Marching a Level Sets.



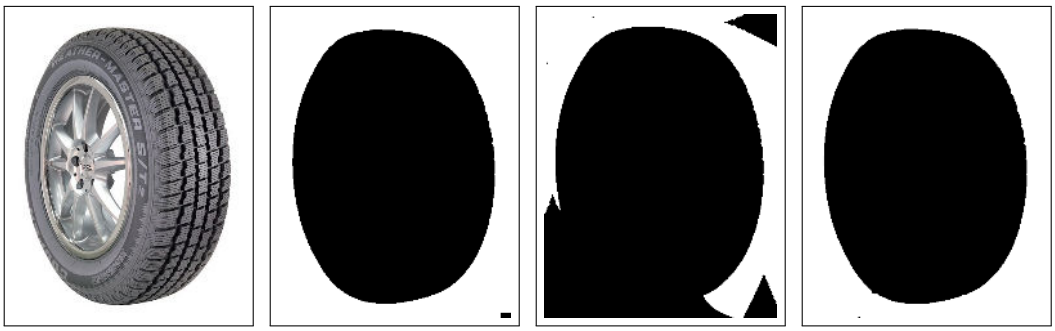
(a) Originál (b) 1 664 ms (c) 146 ms (d) 128 ms

Obr. 27: Výsledok segmentácie malého obrázku s použitím jedného vstupného bodu. *(Zdroj: Autor)*



(a) Originál (b) 7 189 ms (c) 913 ms (d) 872 ms

Obr. 28: Výsledok segmentácie stredného obrázku s použitím jedného vstupného bodu. *(Zdroj: Autor)*



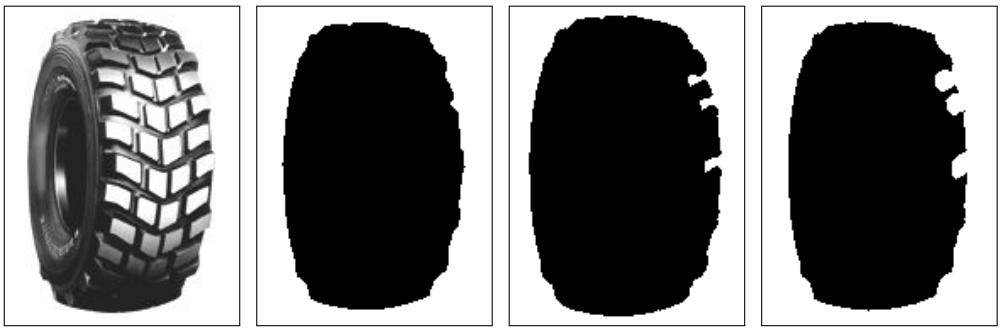
(a) Originál (b) 49 901 ms (c) 4 135 ms (d) 10 709 ms

Obr. 29: Výsledok segmentácie veľkého obrázku s použitím jedného vstupného bodu. *(Zdroj: Autor)*

8.3 Segmentovanie implicitnými kontúrami so štyrmi vstupnými bodmi

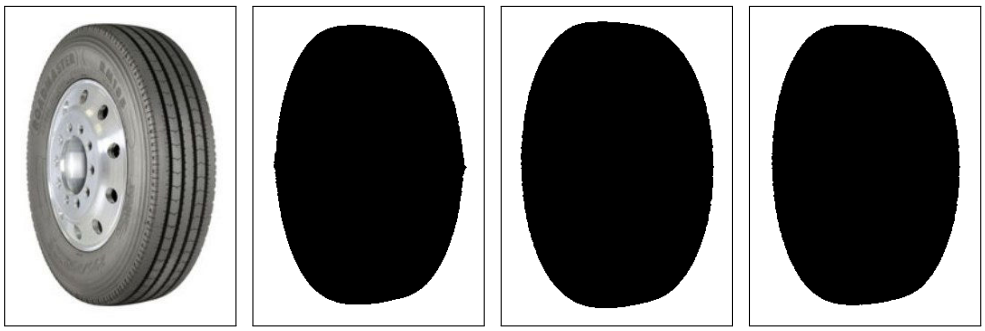
Pri tejto segmentácii boli použité štyri vstupné body, pričom boli rozmiestnené v rohoch obrázku, konkrétne 5px od jeho okrajov. Jednotlivé výsledky sú zoradené rovnako ako pri segmentácií s jedným vstupným bodom a teda:

- **(a)** - je pôvodný obrázok,
- **(b)** - výsledok segmentácie použitím Level Sets algortimu,
- **(c)** - výsledok segmentácie použitím Fast Marching algortimu,
- **(d)** - výsledok segmentácie použitím kombinácie Fast Marching a Level Sets.



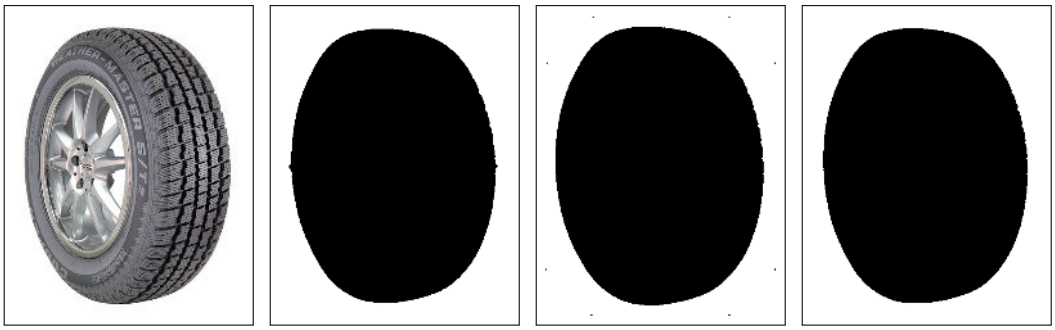
(a) Originál (b) 360 ms (c) 115 ms (d) 141 ms

Obr. 30: Výsledok segmentácie malého obrázku s použitím štyroch vstupných bodov. *(Zdroj: Autor)*



(a) Originál (b) 1 741 ms (c) 909 ms (d) 908 ms

Obr. 31: Výsledok segmentácie stredného obrázku s použitím štyroch vstupných bodov. *(Zdroj: Autor)*



(a) Originál (b) 6 131 ms (c) 7 524 ms (d) 6 776 ms

Obr. 32: Výsledok segmentácie veľkého obrázku s použitím štyroch vstupných bodov. *(Zdroj: Autor)*

8.4 Výsledky testu

Teraz si vyhodnotíme výsledky testu. Nasledujúca tabuľka nám poskytuje informácie o priemerných časoch výpočtu jednotlivých algoritmov na jednotlivých veľkostiach obrázkov. Tieto časy sú udávané v milisekundách.

Tabuľka 5: Tabuľka priemerných časov priebehu výpočtov jednotlivých algoritmov. *(Zdroj: Autor)*

	Malé	**Stredné**	**Veľké**
Snakes	62 ms	257 ms	893 ms
Level Sets	841 ms	5 210 ms	33 901 ms
Fast Marching	109 ms	586 ms	3 384 ms
Fast Marching + Level Sets	100 ms	593 ms	7 398 ms
Level Sets (4 vstupy)	282 ms	1 283 ms	5 210 ms
Fast Marching (4 vstupy)	103 ms	570 ms	4 919 ms
Fast Marching - Level Sets (4 vstupy)	99 ms	592 ms	4 941 ms

Ako môžeme vidieť v tabuľke, najrýchlejší z algoritmov je Snakes. Čas výpočtu narastá s narastajúcou vzdialenosťou kontúry od objektu. Akonáhle by sme mali kontúru v rovnakej vzdialenosti od objektu pri všetkých veľkostiach obrázku, výpočtový čas by narastal pomalšie. Dôvod, prečo by čas neostal rovnaký je jednoduchý. S narastajúcou veľkosťou kontúry narastá aj počet bodov v kontúre a ako už bolo vysvetlené vyššie, každý bod kontúry musí byť prepočítaný. Ako môžeme vidieť na obrázkoch v kapitole 8.1, výsledky sú uspokojivé. Tento algoritmus je vhodný na takéto jednoduché segmentácie kontúr trasologických objektov. Jedine pokiaľ sú objekty umiestnené v strede obrázku a ako výsledok nie je potrebný objekt, ale postačuje hrana objektu.

Fast Marching sa môže zdať o niečo rýchlejší pokiaľ berieme v úvahu jeden vstupný bod, ale aj to má svoje obmedzenia. Táto výhoda platí len do určitej veľkosti segmentovaného objektu. Kvalita výsledku sa narastajúcou veľkosťou zhoršuje až výsledok je neúplný. To môžeme vidieť na obrázku 29(c). Pridaním vstupných bodov sa kvalita výsledku výrazne zlepší, ale čas výpočtu narastie.

Level Sets naproti tomu je výpočtovo náročnejší, pokiaľ berieme v úvahu jeden vstupný bod. Vzhľadom na výpočtovú náročnosť je kvalita výsledkov dobrá aj pri jednom vstupom bode. Pridaním ďalších bodov čas výpočtu rapídne klesne.

Kombináciou týchto algoritmov získame relatívne stabilné prostredie, ktoré je pomerne rýchle, kvalita výsledkov je dobrá a výpočtový čas nezvykne zachádzať do extrémov.

Avšak nič na svete nie je dokonalé, a preto v ďalšej kapitole si povieme, akými vylepšeniami je možné program upraviť tak, aby sme z daných algoritmov vyťažili čo najlepšie výsledky.

9 Návrh vylepšení

Vzhľadom na fakt, že teória okolo implementovaných algoritmov je náročná na pochopenie a rozsah práce nedovoľuje ísť do takej hĺbky. Návrhy vylepšení sa tak týkajú prevažne implementácie. Majme na pamäti, že implementácia je určená prevažne na segmentáciu trasologických objektov na bielom pozadí, a preto sa naše návrhy vylepšení zameriavajú prioritne na túto problematiku.

Prvým vylepšením je možnosť doplnenia pixelov pri implementácii Snakes algoritmu. Ako už bolo spomenuté viackrát, tento algoritmus má problém segmentovať objekty umiestnené na okraji obrázku. Riešením tohto problému je dať užívateľovi možnosť doplniť biele riadky alebo stĺpce podľa potreby. Problém nastane pri komplexnejších obrázkoch, avšak aj tento problém sa dá vyriešiť a to tak, že používateľ podľa potreby dokreslí riadky alebo stĺpce.

Ďalšie vylepšenie sa týka Level Sets algoritmu, pri ktorom sme si ukázali jeho časovú aj výpočtovú náročnosť, a taktiež Fast Marching algoritmu, ktorý nám niekedy dáva zlé výsledky. Jedným z faktorov, ktoré ovplyvňujú čas výpočtu a kvalitu výsledku je aj veľkosť plochy okolo samotného objektu. Pokiaľ by sme chceli segmentovať množstvo obrázkov, napríklad všetky v danom priečinku, určite by sme nechceli segmentovať obrázky po jednom a do každého obrázka zadávať počiatočný bod. V takom prípade je vhodné zvoliť jeden počiatočný bod pre všetky obrázky, napríklad 5 pixelov od ľavého horného rohu. To ešte nerieši problém časovej náročnosti výpočtu. Tak ako sme v predchádzajúcom vylepšení pridávali riadky, respektíve stĺpce, tentokrát navrhujeme ich postupne odstraňovať až do momentu, kedy sa dostaneme do dostatočnej blízkosti objektu. Veľkosť okolia objektu sa tak zníži, čo nám ušetrí čas.

Jedným z vylepšení je prispôsobenie veľkosti obrázku. Pokiaľ je obrázok väčší než je potrebné, namiesto zobrazenia posuvníkov sa obrázok prispôsobí veľkosti kontajneru určeného na zobrazovanie obrázkov. Pokiaľ je obrázok menší, nič sa nezmení. Na takéto prispôsobenie obrázkov sme mysleli už pri vytváraní implementácie, a tak je jeho súčasťou už od počiatku. Napríklad, ak otvoríme obrázok, ktorý má rozlíšenie 2560x1440px a kontajner má rozlíšenie 896x674px, načítaný obrázok je prispôsobený veľkosti kontajnera so zachovaním pomeru strán a teda výsledná veľkosť obrázku je 896x504px.

Náš program sme naprogramovali tak, aby sme na ňom ukázali funkčnosť a použitie jednotlivých algoritmov na segmentáciu trasologických objektov. Avšak to, že tieto vylepšenia momentálne nie sú implementované do programu neznamená, že niekedy v budúcnosti tomu tak nebude.

Záver

Hlavným cieľom bolo vytvorenie implementácií algoritmov aktívnych kontúr na segmentáciu obrazu a navrhnúť vylepšenia. V práci sme sa konkrétne venovali algoritmom Snakes, Level Sets a Fast Marching. Tieto algoritmy sme otestovali na segmentáciu trasologických objektov, konkrétne na segmentáciu dezénu pneumatík.

Prvým čiastkovým cieľom bolo navrhnúť funkcionalitu programu. Týmto cieľom sme získali podrobnú predstavu o funkčnosti programu a teda, čo všetko bude program používateľovi ponúkať. Ďalším krokom bolo vytvorenie architektúry programu, aby boli dodržané určité programátorské praktiky. Vytvorením architektúry sme získali potrebné informácie o štruktúre kódu, ktorý bol napísaný v objektovo orientovanom programovacom jazyku Java. Po získaní týchto informácií sme mohli vytvoriť grafické rozhranie, práve vďaka ktorému môže používateľ ovládať segmentáciu a získavať tak potrebné výsledky.

Keď sme mali úspešne vytvorený základ programu, prešli sme na jednotlivé algoritmy, ktoré sme chceli v našom programe mať implementované. Avšak, pred samotným implementovaním sme si vysvetlili ako dané algoritmy fungujú, aké sú potrebné vstupné parametre a ako získať údaje potrebné pre správne fungovanie algoritmov. Akonáhle sme vedeli, ako algoritmy fungujú a čo potrebujú pre svoje fungovanie, začali sme ich postupne implementovať do nášho programu, pričom sme ich vyskúšali na segmentáciu rôznych objektov umiestnených v rôznych častiach obrázku. Týmto sme zistili nedostatky jednotlivých algoritmov, a to nám pomohlo navrhnúť test, ktorý je spravodlivý ku všetkým algoritmom.

Test pozostával z troch krokov a rozdelil algoritmy do jednotlivých kategórií. Z výsledkov testu tak vieme porovnať algoritmy implicitných a algoritmy explicitných kontúr. Taktiež vieme porovnať, ako sa algoritmy implicitných kontúr správajú pri jednom vstupnom bode a ako sa správajú, keď tieto body začneme pridávať. V teste sme porovnali ako čas výpočtu jednotlivých algoritmov, tak aj kvalitu ich výsledkov. Z výsledkov testu sme dospeli k záveru, za akých okolností je vhodné použiť daný algoritmus a taktiež sme si povedali, prečo výsledky testu dopadli tak, ako dopadli.

V poslednej časti sme sa venovali vylepšeniam nášho programu. Tu sme vymenovali najväčšie nedostatky nášho programu, na ktoré sme navrhli riešenia a poskytli tak čitateľovi prípadnú možnosť upraviť program podľa potrieb. Nie všetky z týchto vylep-

šení sú implementované v programe, avšak to nevylučuje, že niekedy v budúcnosti sa jeho súčasťou nestanú.

Záverom chceme vyjadriť, že táto práca svojimi výsledkami a závermi prispeje k úspešnej realizácií projektu *„Automatizované spracovávanie trasologických objektov"*.

Zoznam bibliografických odkazov

[1] SOBŎTKA, M.; 2012. *Segmentácia obrazu s použitím metód distribuovaných agentov pre získavanie informácie z lekárskych snímkov* [online]. 2012. [cit.5.3.2017] Dostupné na internete: <http://www.itspy.cz/wp-content/uploads/2014/05/acmspy2012_submission_06.pdf>

[2] 2016. *Thresholding (image processing)* [online]. 2016. [cit.5.3.2017] Dostupné na internete: <https://en.wikipedia.org/wiki/Thresholding_(image_processing)>

[3] 2017. *Edge detection methods for finding object boundaries in images* [online]. 2017. [cit.5.3.2017] Dostupné na internete: <https://www.mathworks.com/discovery/edge-detection.html>

[4] GOMES, D.A.; 2014. *Computação Visual e Multimédia* [online]. 2014. [cit.5.3.2017] Dostupné na internete: <http://www.di.ubi.pt/~agomes/cvm/teoricas/07-regionsegmentation.pdf>

[5] 2017. *Image segmentation* [online]. 2017. [cit.5.3.2017] Dostupné na internete: <https://en.wikipedia.org/wiki/Image_segmentation>

[6] 2003. *Digital Image Processing* [online]. 2003. [cit.5.3.2017] Dostupné na internete: <http://user.engineering.uiowa.edu/~dip/lecture/Segmentation3.html>

[7] SETHIAN, J.A.; 2017. *Level Set Methods: An initial value formulation* [online]. 2017. [cit.10.3.2017] Dostupné na internete: <https://math.berkeley.edu/~sethian/2006/Explanations/level_set_explain.html>

[8] FISHER, R.; PERKINS, S.; WALKER, A.; WOLFART, E.; 2003. *Sobel Edge Detector* [online]. 2003. [cit.10.3.2017] Dostupné na internete: <http://homepages.inf.ed.ac.uk/rbf/HIPR2/sobel.htm>

[9] SURI, J.; FARAG, A.; 2007. Deformable Models. *Topics in Biomedical Engineering. International Book Series* [online]. 2007. [cit.10.3.2015] Dostupné na internete: <https://books.google.sk/books?id=CQGwK5f-lboC\&pg=PA33\&lpg=

PA33\&dq=DISTANCE+TRANSFORM+ALGORITHMS+AND\\+THEIR+IMPLEMENTATION+AND+EVALUATION\&source=bl\&ots=rESJ0qiWR\\O\&sig=eKtb8I_S2dgDOj0glLm0sWUAi04\&hl=sk\&sa=X\&ei=DVdaVeKHCYfTU-mvgfAF\&ved=0CDgQ6AEwAw\#v=onepage\&q\&f=false> ISBN 978-0-387-31201-9, s. 33-60

[10] KASS, M.; WITKIN, A.; TERZOPOULOS, D.; 1988. Snakes: Active contour models. In *International Journal of Computer Vision*. [online]. 1988. vol. 1, no. 3 [cit. 13.3.2017] Dostupné na internete: <http://www.cs.ucla.edu/~dt/papers/ijcv88/ijcv88.pdf>. ISSN 1573-1405, s. 321-331

[11] OSHER, S.; SETHIAN, J.A.; 1988. *Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton–Jacobi Formulation* [online]. 1988. [cit. 13.3.2017] Dostupné na internete: <https://math.berkeley.edu/~sethian/2006/Papers/sethian.osher.88.pdf>.

[12] MULDER, W.; OSHER, S.; SETHIAN, J.A.; 1990. *Computing interface Motion in Compressible Gas Dynamics* [online]. 1990. [cit. 13.3.2017] Dostupné na internete: <http://aniso.citg.tudelft.nl/Private/Papers/Journal/JCP100_MOS.pdf>.

[13] SETHIAN, J.A.; 1988. *Fast Marching Methods and Level Set Methods for Propagating Interfaces* [online]. 1988. [cit. 13.3.2017] Dostupné na internete: <https://math.berkeley.edu/~sethian/2006/Papers/sethian.vonkarman_1.pdf>.

[14] SETHIAN, J.A.; 1988. *Advancing Interfaces: Level Set and Fast Marching Methods* [online]. 1988. [cit. 13.3.2017] Dostupné na internete: <https://math.berkeley.edu/~sethian/2006/Papers/sethian.iciamproceedings.1999.pdf>.

[15] SETHIAN, J.A.; 1995. *A Fast Marching Level Set Method for Monotonically Advancing Fronts* [online]. 1995. [cit. 12.3.2017] Dostupné na internete: <https://math.berkeley.edu/~sethian/2006/Papers/sethian.fastmarching.pdf>.

Zoznam príloh

Príloha A: Systémová dokumentácia

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

POKROČILÉ METÓDY DETEKCIE KONTÚR A ICH POUŽITIE PRI ROZPOZNÁVANÍ TRASOLOGICKÝCH OBJEKTOV

Systémová dokumentácia

Banská Bystrica 2017 **Bc. Marcel Koč**

Obsah

1 Úvod

Táto systémová dokumentácia je venovaná všetkým, ktorých zaujíma, akým spôsobom je navrhnutá aplikácia na implementačnej úrovni, za účelom jej preštudovania alebo vylepšenia. Aplikácia je vydaná pod GNU GPL licenciou, takže jej zdrojové kódy sú voľne dostupné a môže ich ktokoľvek upravovať a šíriť. K aplikácií sa nevzťahuje žiadna záruka a v prípade ďalšieho šírenia musí byť distribuovaný pod GNU GPL licenciou.

2 Štruktúra súborov

Hrubým textom sú označené priečinky a kurzívou súbory.

- *Main.java*
- **gui**
  - **components**
    - *Console.java*
    - *FM.java*
    - *FMLS.java*
    - *ImageContainer.java*
    - *LS.java*
    - *SidePanel.java*
    - *SNAKE.java*
    - *TopPanel.java*
  - *ImageMouseListener.java*
  - *ImageMouseMotionListener.java*
  - *Listeners.java*
  - *MainWindow.java*

- **algorithm**
  - *Algorithms.java*
  - *BandElement.java*
  - *BandElementCache.java*
  - *Coordinate.java*
  - *DeferredArray3D.java*
  - *DeferredByteArray3D.java*
  - *DeferredDoubleArray3D.java*
  - *DeferredIntArray3D.java*
  - *DeferredObjectArray3D.java*
  - *FastMarchingAlgorithm.java*
  - *ChamferDistance.java*
  - *ImageObject.java*
  - *LevelSetsAlgorithm.java*
  - *SharedSpace.java*
  - *SnakesAlgorithm.java*
  - *States.java*

3 Po spustení aplikácie

Aplikácia je v programovacom jazyku *Java* a hlavná trieda nesie názov *Main*. Účelom tejto triedy je vytvoriť inštanciu grafického rozhrania a zobraziť ju. Jadro grafického rozhrania sa nachádza v triede *MainWindow*, ktorá vytvára inštancie objektov grafických komponentov. Každý grafický komponent má vlastnú triedu, ktorá inicializuje grafické prvky a metódy potrebné pre získavanie a nastavovanie potrebný údajov.

4 Po spustení segmentácie

Pokiaľ je celé grafické rozhranie inicializované a zobrazené, aplikácia je schopná používania. Stlačením tlačidla *Štart* sa vytvorí nové vlákno, ktoré inicializuje triedu *SharedSpace*, ktorej parametrom je grafické rozhranie a zavolá funkciu na spustenie segmentácie. *SharedSpace* obsahuje dve základné funkcie, ktoré spusťajú segmentáciu. Prvá funckia *startSegmentation*, je zavolaná ak sa segmentuje iba jeden obrázok a jej účelom je vytvorenie inštancie algoritmu, ktorý segmentuje daný obrázok a spustenie daného algoritmu. Ďalšia funkcia *multiSegmentation* sa zavolá pokiaľ bol do aplikácie načítaný priečinok. Úlohou tejto funkcie je zavolať funkciu segmentácie v cykle, pričom meria časy výpočtu algoritmu a výsledky ukladá do priečinkov, ktoré nesú názvy algoritmov. Práve táto funkcia obstaráva segmentáciu viacerých obrázkov postupne, a tým šetrí čas a námahu používateľa.

Ako sme už vyššie spomenuli, vstupným parametrom triedy *SharedSpace* je grafické rozhranie. Keďže sa táto inštancia nachádza v inom vlákne, je potrebné zaistiť, aby bola prepojená s grafickým rozraním. Grafické rozhranie, ktoré pošleme parametrom do inštancie *SharedSpace* zaručí možnú komunikáciu medzi týmito dvoma objektmi. Okrem toho, grafické rozhranie je aj vstupným parametrom pre jednotlivé inštancie algoritmov. Takto vieme zaručiť, že algoritmus dokáže vykresliť výsledok priamo do grafického komponentu. Úlohou inštancie konkrétneho algoritmu je segmentovať objekty a výsledky posielať grafickému rozhraniu, ktoré ich vykreslí. Ako jednotlivé inštancie algoritmov pracujú sme si už povedali v našej práci, konkrétne v časti implementácií jednotlivých algoritmov.

Príloha B: Používateľská dokumentácia

UNIVERZITA MATEJA BELA V BANSKEJ BYSTRICI
FAKULTA PRÍRODNÝCH VIED

POKROČILÉ METÓDY DETEKCIE KONTÚR A ICH POUŽITIE PRI ROZPOZNÁVANÍ TRASOLOGICKÝCH OBJEKTOV

Používateľská dokumentácia

Banská Bystrica 2017 **Bc. Marcel Koč**

Obsah

1 Úvod

Používateľská príručka je vytvorená na informovanie používateľa o tom, ako správne používať našu aplikáciu, ako sa v nej orientovať a získavať požadované výstupy. Najprv si ukážeme grafické rozhranie aplikácie, ktoré si neskôr podrobnejšie rozpíšeme a dozvieme sa, na čo jednotlivé časti slúžia.

1.1 Spustenie aplikácie

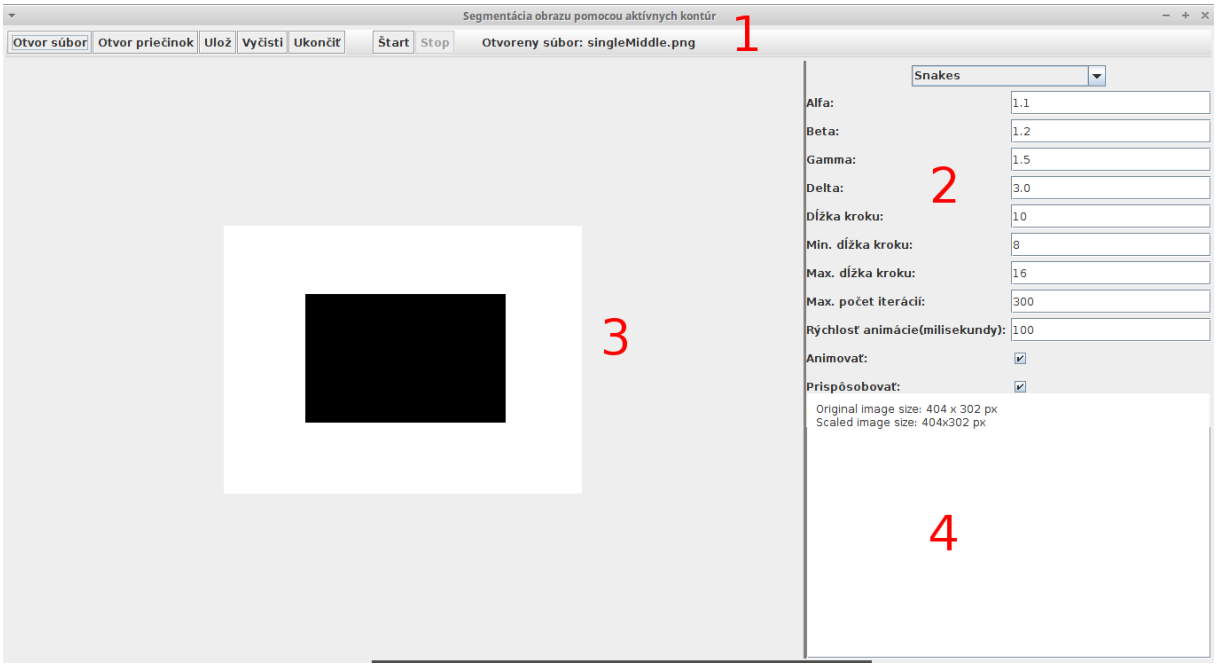
Na to, aby sme mohli spustiť aplikáciu je potrebné mať v počítači nainštalovanú *Javu*. Java je platforma, na ktorej je postavených množstvo systémov, ale súčasne je to aj názov programovacieho jazyka, v ktorom je naša aplikácia naprogramovaná. Pokiaľ chceme aplikáciu spustiť a nemáme v počítači nainštalovanú Javu, je potrebné ju stiahnuť a nainštalovať napríklad odtiaľto: `https://www.java.com/en/download/manual.jsp`. Aplikácia sa spúšťa v závislosti od operačného systému. Pokiaľ je aplikácia spustená pod operačným systémom *Windows*, dvojklikom na ikonku sa aplikácia spustí. Pokiaľ použijeme operačný systém *Linux*, je potrebné spustiť príkazom a to:

```
1 java -jar Snake.jar
```

2 Grafické rozhranie

Grafické rozhranie umožňuje komunikáciu používateľa s aplikáciou. Vďaka tomuto rozhraniu je možné nahrávať obrázky, sledovať priebeh algoritmu, poprípade uložiť výsledok segmentácie. Grafické rozhranie je rozdelené na štyri základné časti a to na:

1. panel možností,
2. vstupné parametre,
3. kontajner s obrázkom,
4. konzola.



V nasledujúcich podkapitolách si podrobnejšie rozoberieme jednotlivé časti, na čo nám služia a čo dokážu ovplyvniť.

2.1 Panel možností

V tomto panely sa nachádza niekoľko tlačidiel umiestnených v jednom riadku, na konci ktorého je textové pole s názvom obrázku, respektíve priečinku. Teraz si vysvetlíme, čo jednotlivé tlačidlá ovplyvňujú.

- **Otvor súbor** je tlačidlo, po ktorého stlačení sa používateľovi zobrazí dialógové okno a vyzve používateľa, aby vybral jeden obrázok. Tento obrázok bude vstupným obrázkom určeným k segmentácií objektov.
- **Otvor priečinok** slúži na zvolenie daného priečinku. Narozdiel od predošlého tlačidla, táto voľba vyberá priečinok, pričom na vstupe do segmentácie nie je jeden obrázok, ale všetky obrázky v danom priečinku. Touto voľbou dokážeme segmentovať viacero obrázkov, a tým uľahčiť prácu používateľovi.
- **Ulož** slúži na ukladanie výstupov do zvoleného priečinka.
- **Vyčisti** slúži na vymazanie všetkých vstupných údajov okrem obrázku. Stlačením tohto tlačidla sa vymažú všetky už zadané vstupné body, a tak sa pôvodný obrázok

zresetuje. V prípade, že používateľ zadá zlé vstupné body a bude ich chcieť vymazať alebo bude chcieť vymazať výsledky segmentácie, toto tlačidlo to zabezpečí.

- **Ukončiť** aplikáciu.
- **Štart** tlačidlo slúži na spustenie výpočtu algoritmu. Týmto tlačidlom sa vytvorí inštancia daného algoritmu, ktorého priebeh môžeme vidieť v kontajneri s obrázkom. Stlačením tohto tlačidla sa automaticky zablokuje, čím sa zabezpečí, aby výpočet nebol spustený viackrát.
- **Stop** tlačidlo je zablokované, avšak vieme ho aktivovať spustením algoritmu. Funkcia tohto tlačidla je dostupná len počas priebehu algoritmu. Stlačením sa nie len algoritmus zastaví, ale súčasne sa odblokuje tlačidlo *Štart*.

2.2 Vstupné parametre

Tento komponent slúži na výber algoritmu, ktorý bude použitý pri segmentácií. Algoritmus je možné zvoliť výberom v hornej časti panelu, pričom sa pod zvoleným algoritmom zobrazia parametre potrebné pre daný algoritmus. To, čo dané parametre ovplyvňujú sme si spomenuli v našej práci, v časti implementácií algoritmov.

2.3 Kontajner s obrázkom

Kontajner s obrázkom slúži len na zobrazovanie vstupného obrázku a zadávanie vstupných bodov, respektíve kontúry. Po spustení algoritmu sa práve v tomto komponente zobrazuje priebeh výpočtu a po jeho ukončení aj výsledok.

2.4 Konzola

Úlohou konzoly je informovať užívateľa o tom, čo sa v danom momente deje. Tu sa zobrazia informácie o rozmeroch vstupného obrázku, kedy je algoritmus spustený a kedy ukončený. Taktiež sa tu zobrazí čas výpočtu.

Príloha C: Sprievodné CD so zdrojovým kódom