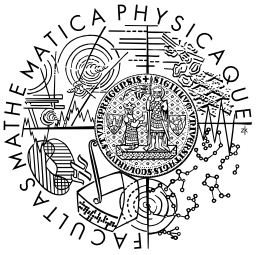


MATEMATICKO-FYZIKÁLNÍ FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Jakub Kopál

Určování ostrosti a segmentace obrazu

Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: doc. RNDr. Elena Šikudová, Ph.D.

Studijní program: Informatika

Studijní obor: ISDI

Praha 2018

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V ..... dne ..... Podpis autora

Ďakujem svojej vedúcej doc. RNDr. Elene Šikudovej, Ph.D. za venovaný čas a užitočné rady pri vývoji programu a vypracovaní bakalárskej práce.

Název práce: Určování ostrosti a segmentace obrazu

Autor: Jakub Kopál

Katedra: Katedra softwaru a výuky informatiky

Vedoucí bakalářské práce: doc. RNDr. Elena Šikudová, Ph.D., Katedra softwaru a výuky informatiky

Abstrakt: Problém automatickej segmentácie obrazu sa ukázal byť zložitý a dodnes nie zcela vyriešený. Kedže sa jedná o tak komplexný problém, táto práca sa ho nepokúša riešiť vo svojej najobecnejšej podobe. Namiesto toho sa zameriava na automatickú, binárnu segmentáciu obrazu, s možnosťou zvoliť rôzne atribúty podla ktorých sa má segmentácia vykonávať. Medzi tieto atribúty patrí ostrosť a farba obrazu. Výsledky segmentácie založenej na predpoklade "zaostrený objekt, nezaostrené okolie" sa ukázali byť velmi podobné groundtruth na obrázkoch ktoré tento predpoklad spĺňajú.

Klíčová slova: Grayscale, GroundTruth, Segmentácia, Pravdepodobnosť

Title: Local sharpness prediction and image segmentation

Author: Jakub Kopál

Department: Department of Software and Computer Science Education

Supervisor: doc. RNDr. Elena Šikudová, Ph.D., Department of Software and Computer Science Education

Abstract: The problem of automatic segmentation turned out to be complicated and to this day, not completely solved. Since it is a complex problem, this paper is not trying to solve it in its most general form. Instead, it is focused on automatic, binary picture segmentation, with the option to choose attributes, based on which the segmentation should operate. Among these attributes are the focus and color of the picture. The results of the segmentation based on the assumption "focused object, blurry background" turned out to be very similar to the groundtruth in pictures, which fulfill this assumption.

Keywords: Grayscale, GroundTruth, Segmentation, Probabilities

Obsah

Úvod

Kontext

Problém automatického rozpoznávania obrazu je už dlhé roky nevyriešenou úlohou, ktorej vyriešenie sa ukázalo byť veľmi zložité. Tento problém je predmetom mnohých odborných diskusií a mnoho ľudí po celom svete sa tomuto problému odborne venuje. Z týchto dôvodov sa táto práca problém automatickej segmentácie obrazu nepokúša riešiť vo svojej najobecnejšej podobe. Namiesto toho sa budeme zameriavať len na binárnu segmentáciu. O segmentovaných obrázkoch budeme mať navyše určité predpoklady. S týmito predpokladmi budeme pracovať tak, aby sme dosiahli rozumnú úroveň správnosti segmentácie.

Množstvo fotiek je vytváraných s úmyslom zachytiť objekt(zviera, športovca, auto ...). Pokiaľ sa objekt podarí dobre zachytiť, je zaostrený, zatiaľ čo okolité prostredie je nezaostrené. Keďže tento invariant platí pre množstvo fotografii, bude naša segmentácia založená práve na ňom.

Cieľ

Naším pôvodným cieľom bolo implementovať software, ktorý by rozoznával zaostrené a nezaostrené oblasti na fotografiách. Tento cieľ sme neskôr rozšírili na binárnu segmentáciu obrazu. Segmentácia obrazu sa dá rozdeliť na dva hlavné prístupy : automatickú a interaktívnu.

Automatická segmentácia. Obecná automatická segmentácia, ktorá o segmentovanom obraze nič nepredpokladá, sa ukázala byť príliš zložitá. Z tohto dôvodu budeme o segmentovanom obraze mať určité predpoklady. Podľa toho aké predpoklady očakávame, sa nám automatická segmentácia rozložila do troch častí.

- segmentácia podľa ostrosti (hlavný cieľ)
- segmentácia podľa farby
- segmentácia podľa ostrosti a farby.

Pôvodným cieľom bola len automatická segmentácia podľa ostrosti. Až v priebehu vývoja vznikla myšlienka experimentálne vyskúšať aj dve zvyšné varianty pomocou nástrojov, ktoré sme mali v tom čase už prevažne implementované. **Interaktívna segmentácia**. Interaktívna segmentácia požaduje od užívateľa určitú pomoc pri segmentovaní, vyznačením úzenia s daným významom. Aj keď by interaktívna segmentácia nemusela mať o segmentovanom obraze žiadne predpoklady,

naším cielom je interaktívnou pomocou zlepšiť výsledky automatickej segmentácie. Z tohto dôvodu aj pre obrazy segmentované interaktívne požadujeme jeden z troch vyššie uvedených predpokladov.

Štruktúra práce

Prvá kapitola popisuje možné postupy na dosiahnutie nášho cieľa, ako aj odôvodnenie prečo bol zvolený výsledný postup.

V druhej kapitole je detailne vysvetlený použitý postup celého procesu segmentácie. Zahŕňa postupy odvodené z vedeckej literatúry, ako aj postupy a úpravy založené na vlastných heuristikách a pozorovaniach. Analyzuje možné prístupy k riešeniu vzniknutých problémov a porovnáva ich efektivitu.

Tretia kapitola obsahuje programátorskú dokumentáciu. Tu sa popisuje predovšetkým schéma objektového návrhu spolu s technickými vlastnosťami programu.(použité technológie, jazyky, konvencie ...)

Piata kapitola stručne popisuje používateľské rozhranie, a ilustruje príklady použitia.

Šiesta kapitola sa venuje porovnaniu našich výsledkov s existujúcimi riešeniami rovnakého problému. Dataset tridsiatich obrázkov, splňujúcich náš invariant, bol ručne rozsegmentovaný a následne porovnaný a výsledkami nášho programu a programom Adobe Photoshop. V tejto kapitole sa na výsledkoch ilustrujú vyhody aj slabiny nášho riešenia.

Posledná kapitola sa venuje zhrnutiu výsledkov ako aj práce samotnej.

1. Analýza

Táto kapitola je venovaná analýze problému segmentácie. Táto kapitola sa však venuje len rozhodnutiam, ktoré museli byť učinené na začiatku vývoja (zmapovanie odbornej literatúry venujúcej sa danej problematike). V tomto štádiu sme neboli uvedomený o všetkých problémoch ktoré vzniknú počas implementácie. Analýze týchto problémov sa venuje kapitola 2.

1.1 Čas výpočtu

Naším cieľom je možnosť segmentovať aj obrázky s vysokým rozlíšením(10 megapixelov). Kedže budeme pracovať s ohromným počtom pixelov, musíme od celého procesu segmentácie požadovať lineárnu časovú zložitosť vzhladom na počet pixelov. Polynomiálna časová zložitosť by pre 10 megapixelové fotografie urobila náš program nepoužiteľným. Ďalšou kľúčovou voľbou je výber programovacieho jazyka. Zvolený jazyk by mal umožňovať vytvárať abstrakciu nad kódom(oop), ale predovšetkým čo najväčšiu možnú optimalizáciu kódu pre konkrétny procesor. Kvôli jitovaniu kódu počas runtimu, intenzívnej práce s poľami(range check by viedol k spomaleniu) a slabším optimalizáciam, jazyky ako C# a Java neboli zvolené. Strojový kód generovaný z jazyka C je síce vysoko optimalizovaný, avšak C nie je objektovo orientovaný jazyk, a tým neumožňuje potrebný level abstrakcie. Jazyk, ktorý spĺňa obe naše požiadavky je C++. Z týchto dôvodov bol na implementáciu samotnej segmentácie zvolený práce tento jazyk.

1.2 Rozšíritelnosť

Našim cieľom je implementovať program, ktorý obsahuje len jednu konkrétnu funkciu(segmentovať obraz). Z tohto dôvodu nie je vhodné vytvárať finálnu aplikáciu(napr. konzolovú aplikáciu), ale použiť knižnicu s jednou exportovanou funkciou vykonávacou segmentáciu. Týmto zaistíme možnosť rozširovať iné komplexnejšie aplikácie o funkcionalitu našej knižnice.

1.3 Existujúce riešenia

1.3.1 Štruktúra výpočtu

Skôr ako prejdeme na existujúce riešenia, je dôležité rozmyslieť si vhodnú štruktúru výpočtu. Našim prvotným cieľom bola automatická binárna segmentácia podľa ostrosti, s možnosťou doplnenia o interaktívnu pomoc. Celý proces si tak rozdelíme na dve hlavné časti.

Cieľom prvej časti bude každému pixelu priradiť pravdepodobnosť [0,1] tak že:

$$p(x) = 1 \Leftrightarrow \text{pixel x je je určite z popredia}$$

$$p(x) = 0 \Leftrightarrow \text{pixel x je je určite z pozadia}$$

Neskôr sme si proces získavania pravdepodobností rozšírili o dve varianty(farba, ostrosť a farba). Proces získavania týchto pravdepodobností bude pre každý z troch typov automatickej a interaktívnej segmentácie odlišný.

Cieľom druhej časti bude každý pixel priradiť buď do pozadia, alebo popredia na základe výsledkov z prvej časti. Druhú časť už nebude zaujímať o aký konkrétny typ segmentácie sa jedná, ani akým spôsobom boli tieto pravdepodobnosti vypočítané.

1.3.2 Detekcia ostrosti

Detekcia ostrosti sa venuje prvej časti automatickej segmentácie podľa ostrosti. Cieľom je priradiť každému pixelu priradiť pravdepodobnosť [0,1]. Táto pravdepodobnosť reprezentuje ostrosť daného pixelu. Problematike ostrosti obrazu sa venuje mnoho odbornej literatúry. My sme však požadovali, aby výsledkom bola informácia o ostrosti každého pixelu, nie celého obrazu. Ďalej sme požadovali lineárnu časovú zložitosť v závyslosti od počtu pixelov. Po prechádzaní odbornej literatúry venujúcej sa nášmu problému, sme napokon zvažovali články [1] a [2]. Oba splňovali všetky naše požiadavky a sľubovali dobré výsledky. Napokon ako základ pre implementáciu prvej časti bol vybraný článok [2] vďaka výpočtovo nenáročnému algoritmu. Implementácia podľa článku [1] by však mohla dosahovať rovnako dobré, ak nie lepšie výsledky. Celý proces výpočtu ostrosti, ako aj finálna mapa ostrosti obrazu sa odteraz bude nazývať *S3*, podľa tohto článku.

1.3.3 Interaktívna segmentácia

V našom konkrétnom prípade požadujeme, aby interaktívna segmentácia len zdokonalila výsledky(pravdepodobnosti) automatickej segmentácie. Ďalej predpokladáme, že aj interaktívne segmentovaný obrázok bude spĺňať dopredu určený predpoklad. Z týchto dôvodov sú naše požiadavky na interaktívnu segmentáciu veľmi špecifické a stávajúce riešenie, ktoré by presne spĺňalo naše požiadavky, sme nenašli. Zvolený postup vytvárania modelu nad interaktívne vyznačenými oblasťami, ktorý s výhodou využíva uvedený predpoklad, sme si tak vytvárali sami na základe vlastných heuristík.

1.3.4 Segmentácia

Keď už máme mapu pravdepodobností, musíme každý pixel priradiť popredíu alebo pozadiu. Presne pre tento problém existuje známe riešenie pomocou minimálneho rezu [3]. Riešenie pomocou minimálneho rezu avšak nie je vyhovujúce kvôli svojej nelineárnej časovej zložitosti.

Z tohto dôvodu je segmentácia navrhnutá podľa článku [4], ktorý pomocou časovo efektívnejšieho algoritmu dosahuje podobných výsledkov ako min-cut. Proces segmentácie odteraz budeme nazývať *Geos*.

2. Návrh riešenia

Táto kapitola je venovaná popisu celého procesu segmentácie. Je rozdelená do piatich častí.

1. S3 (Výpočet pravdepodobnostnej mapy podľa ostrosti).
2. Interaktívna segmentácia.
3. S3 & Color (algoritmus S3 rozšírený o farebnú heuristiku).
4. Farebná segmentácia.
5. Geos (finálna segmentácia na základe pravdepodobnostnej mapy).

2.1 S3

Cieľom *S3* algoritmu je priradiť každému pixelu číslo z [0,1] pričom platí

$$p(x) = 1 \Leftrightarrow \text{pixel x je maximálne ostrý}$$

$$p(x) = 0 \Leftrightarrow \text{pixel x je maximálne neostrý}$$

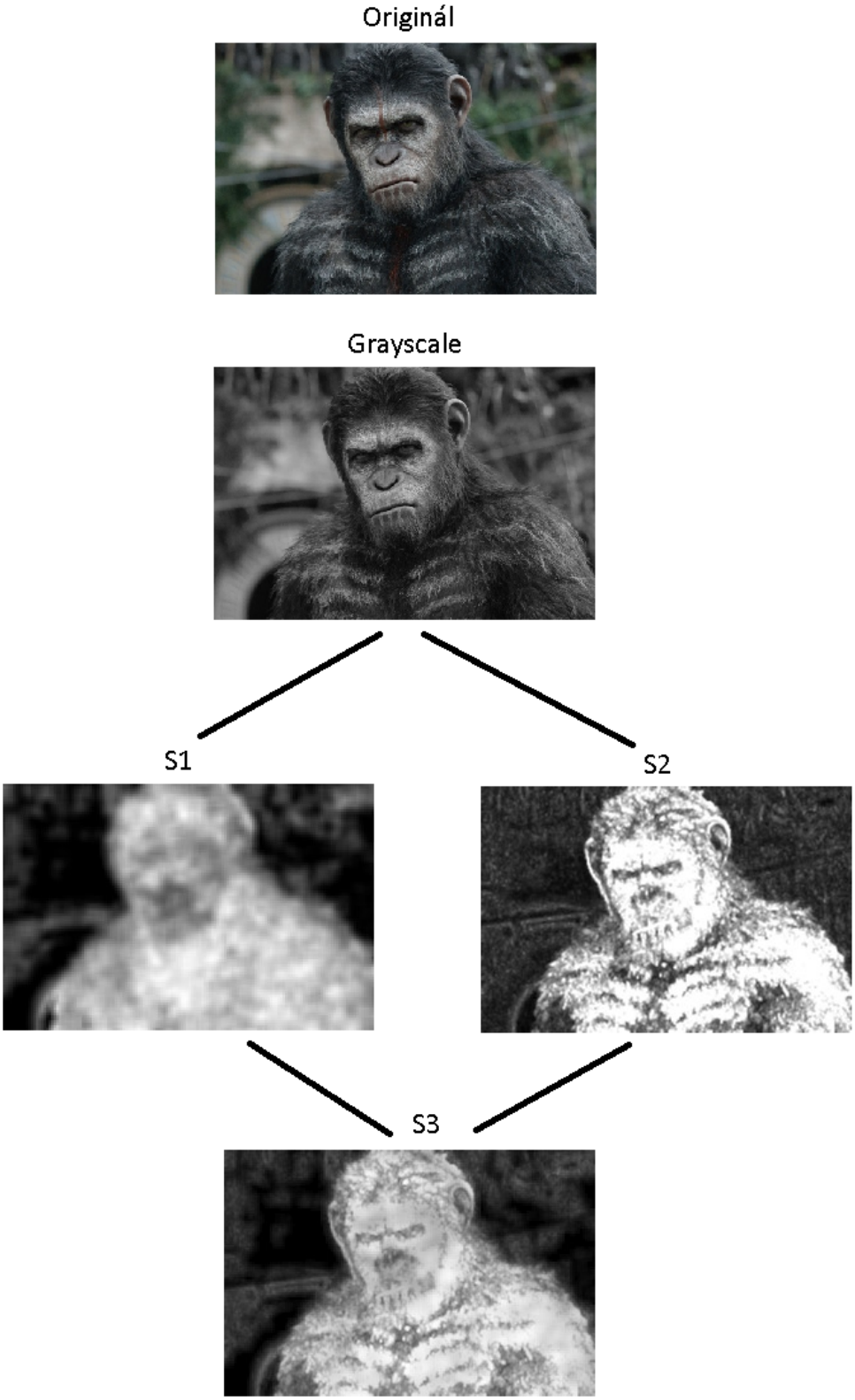
Tieto pravdepodobnosti neskôr poslúžia ako vstup pre samotné binárne segmentovanie. Ako podklad pre výpočet daných pravdepodobností slúži článok [2], ktorého postup je neskôr drobne upravený pre konkrétne potreby. Nasledujúca kapitola sa zameriava predovšetkým na detailný popis zvolenej implementácie a na poukázanie jej výhod. Algoritmus samotný bude vysvetlený len okrajovo. Detaily algoritmu opisuje článok [2].

2.1.1 Štruktúra algoritmu

Algoritmus dostane na vstupe obrázok v podobe poľa pixelov. Každý pixel sa nachádza v RGB formáte. Na výstupe je pole reprezentujúce ostrosť pre každý pixel. Výsledná mapa pravdepodobnosti nazývame *S3*. Na výpočet *S3* budeme potrebovať medzivýsledky *S1* a *S2*. Štruktúra výpočtu vyzerá nasledovne

1. Prevod obrázku z RGB do Grayscale
2. Výpočet *S1* mapy
3. Výpočet *S2* mapy
4. Výpočet *S3* mapy zlúčením *S1* a *S2*

Obe mapy *S1* a *S2* sa počítajú z šedotónovej verzie pôvodného obrázku. Nasledujúce časti popisujú zhotovenie *S1* , *S2* a *S3*. Šedotónové prevedenie pôvodného obrazu budeme nazývať *GrayScale*.



Obr. 2.1: Štruktúra výpočtu

2.1.2 S1

S1 algoritmus rozlišuje oblasti z väčšou a menšou zrnitosťou obrazu. Predpokladá, že zrnité časti obrazu predstavujú ostrosť a naopak oblasti s malou zrnitosťou predurčujú neostrosť.

Algoritmus

Nasledujúca časť popisuje len veľmi hrubý myšlienkový proces algoritmu. Samotné technické detaily sa dočítate [2]. Algoritmus na výpočet *S1* je časovo jednoznačne najnáročnejší, preto pri efektivite implementácie bude kladený najväčší dôraz.

Bloky Pixele obrázka sa rozdelia do blokov 32x32 pixelov. Začiatok každého nasledujúceho bloku je posunutý o 8 pixelov. Kedže ostrosť ako taká v kontexte jedného pixelu nedáva zmysel algoritmus počíta ostrosť v kontexte celého bloku. Cieľom *S1* algoritmu je teda každému bloku určiť jeho ostrosť. Finálna ostrosť pixelu bude nakoniec aritmetický priemer všetkých blokov, ktoré sa nad daným pixelom prekrývajú.

Prefiltrovanie blokov Kedže počítanie ostrosti bloku ako bude ukázané neskôr nie je časovo jednoduchý proces, pokúsime sa triviálne neostré bloky odfiltrovať. Na to budeme potrebovať maximálny, minimálny a priemerný jas pixelu bloku, kde jas pixelu x značíme l(x) a počíta sa ako

$$l(x) = (0.7656 + 0.0364 * g(x))^{2.2}$$

kde g(x) je hodnota pixelu x v grayscale. Ak "kontrast"bloku bude vyhodnotený ako príliš nízky, jeho hodnota sa nadstaví na nulu. Hodnoty maximálneho, minimálneho a priemerného jasu pixelu bloku budeme odteraz značiť skratkou **3m** (ako max, min, mean).

Ostrosť bloku Pre všetky bloky, ktoré sa nepodarilo odfiltrovať treba spočítať skutočnú ostrosť. Tá sa počíta nasledovne

1. Pre blok spočítame 2D Fourierovu transformáciu
2. Pomocou lineárnej regresie odhadneme sklon magnitúdového spektra

Myšlienka daného postupu spočíva v zachytení a zmeraní šumu v bloku. Čím vyšší šum, tým vyššia ostrosť. Výsledok po Fourierovej transformácii bude klesať rýchlejšie od stredu k okrajom pre bloky s menšou ostrosťou. Spád tohto klesania sa pokúsime odhadnúť lineárnou regresiou. Sklon lineárnej funkcie zachycujúcej klesanie transformovaného bloku nám určí finálnu ostrosť bloku. Čím vyšší sklon, tým nižšia ostrosť.

Obr. 2.2: **A)** Blok s nižšou ostrosťou. **B)** Blok s vyššou ostrosťou. **C)** Fourierova transformácia bloku *A*. **D)** Fourierová transformácia bloku *B*.

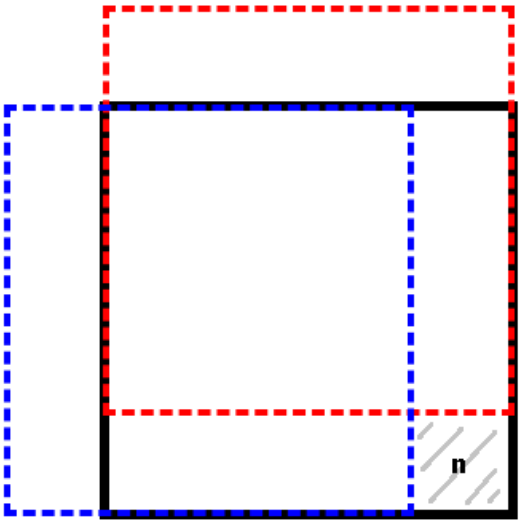
Implementácia

V tejto časti sa detailne popisuje implementácia použitého algoritmu.

Priamočiary prístup Prvý postup, ktorý je okamžite zrejmý je počítanie každého bloku zvlášť. Teda pre každý blok nájsť 3m. Na základe týchto troch hodnôt otestovať podmienku či daný blok prefiltrovať alebo pokračovať vo výpočte.

Analýza problému Je zrejmé, že priamočiary postup bude fungovať a bude implementačne veľmi jednoduchý. Zároveň je však zrejmé, že nebude časovo optimálny kvôli veľkému prekryvu blokov. Obecne sa nad jedným pixelom prekrýva šestnásť blokov. To znamená, že pri počítaní 3m sme každý pixel počítali šestnásť krát. Myšlienka teda je nepočítať všetko nanovo, ale využiť predpočítané hodnoty bloku na výpočet bloku bezprostredne nasledujúceho. Túto myšlienku je možné principiálne použiť len na počítanie 3m, teda na samotné filtrovanie. Fourierová transformácia predchádzajúceho bloku nám nepomôže efektívnejšie dopočítať fourierovú transformáciu nasledujúceho bloku kedže jej obraz závisí na všetkých hodnotách vstupnej funkcie. Z pozorovania vyplýva, že filtrovanie blokov (ktoré sa na rozdiel od fourierovej transformácie počíta pre každý blok) je možné lepším než priamočiarym riešením zrýchliť až šestnásť krát.

Použité riešenie Myšlienka použitého riešenie spočíva vo využívaní predpočítaných výsledkov 3m predchádzajúceho bloku na výpočet týchto hodnôt bloku nasledujúceho ako ilustruje obrázok 2.1.2

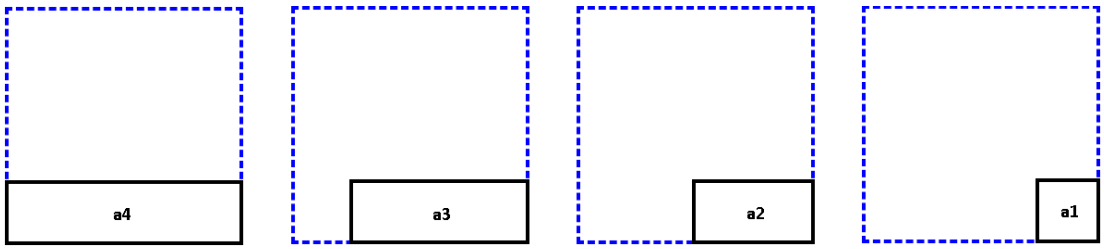


Obr. 2.3: Hodnoty 3m v červenom aj modrom bloku máme výpočítané. Cieľom je dopočítať tieto hodnoty čiernemu bloku, pozeraním sa na pixele len v oblasti *n*.

Štvorce budeme odteraz označovať nasledovne

- Čierny štvorec ako **C** (Current)
- Modrý štvorec ako **L** (Left)
- Červený štvorec ako **U** (Up)

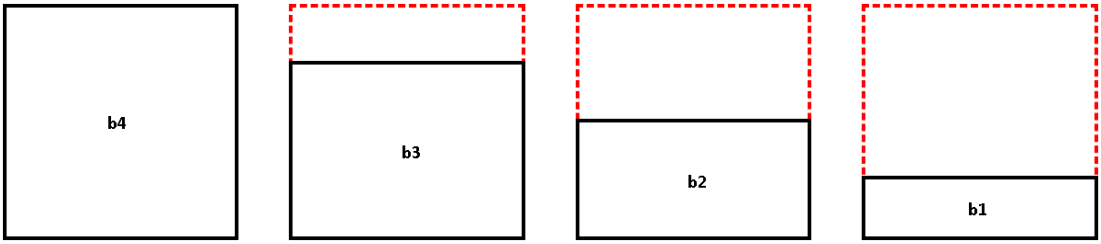
Prvý krok je spočítať 3m pre oblasť *n*. Tá obsahuje 64 pixelov. Z indukčného predpokladu máme hodnoty 3m pre oblasti *a1*, *a2*, *a3* a *a4* spočítané.



Pre zachovanie indukčného predpokladu musíme 3m pre zóny *a1*, *a2*, *a3* a *a4* spočítať aj pre blok **C**. Tie však vieme dopočítať jednoducho

- $C_{a1}.max = n.max$
- $C_{a2}.max = max(C_{a1}.max, L_{a1}.max)$
- $C_{a3}.max = max(C_{a1}.max, L_{a2}.max)$
- $C_{a4}.max = max(C_{a1}.max, L_{a3}.max)$

Obdobne potom aj pre minimum a priemer. Priemer počítame až nazáver zo sumy všetkých pixelov. Demonštrovaný postup ukazuje ako spočítať 3m pre spodnú štvrtinu bloku **C**. Pritom namiesto toho, aby sme 3m počítali zo všetkých 256 pixelov spodnej štvrtiny stačilo prejsť len 64 pixelov a následnými deviatimi porovnaniami sme dopočítali celú spodnú štvrtinu. Podobný princíp použijeme aj pre dopočítanie celého bloku **C** .
Z indukčného predpokladu pre blok **U** máme hodnoty 3m pre oblasti *b1*, *b2*, *b3* a *b4* spočítané.



Obdobne ako v predošlom prípade pre zachovanie indukčného predpokladu musíme 3m pre zóny *b1*, *b2*, *b3* a *b4* spočítať aj pre blok **C**. Tie dopočítame nasledovne

- $C_{b1}.max = C_{a4}.max$
- $C_{b2}.max = max(C_{b1}.max, U_{b1}.max)$
- $C_{b3}.max = max(C_{b1}.max, U_{b2}.max)$
- $C_{b4}.max = max(C_{b1}.max, U_{b3}.max)$

Podobne pre minimum a priemer. Hodnoty C_{b4} sú údaje o 3m pre celý blok a teda náš záverečný výsledok. Princíp prečo pri platnosti indukčného predpokladu platí tento predpoklad aj pre nasledujúce bloky je jasný. Avšak tento predpoklad je potrebné na začiatku naplniť. Typicky pre prvý blok v stĺpci a riadku. Tu sa ponúka priamočiare riešenie a to pre každý prvý blok v riadku/stĺpci spočítať hodnoty 3m pre zóny *a1*, *a2*, *a3*, *a4* *b1*, *b2*, *b3* a *b4* zo všetkých pixelov v týchto zónach. Vďaka zanedbateľne malému počtu týchto blokov (rádovo $\sqrt{n}$ kde n je počet pixelov obrázka) sa strata neprejaví v reálnych výsledkoch. Aj tu sa však dá použiť (a aj je použitá) obmena vysvetleného princípu, len nie v takej miere.

Záverečné porovnanie Každý blok obsahuje 1024 pixelov. Pri priamočiarom postupe na výpočet 3m potrebuje spraviť $3 * 1024$ operácií (každý pixel porovnať pre maximum, minimum a priemer). Pri demonštrovanom postupe dosiahneme rovnakého výsledku pomocou $3 * 64 + 9 + 9$ operacií.

- $3 * 64$ pre výpočet 3m v novej oblasti n

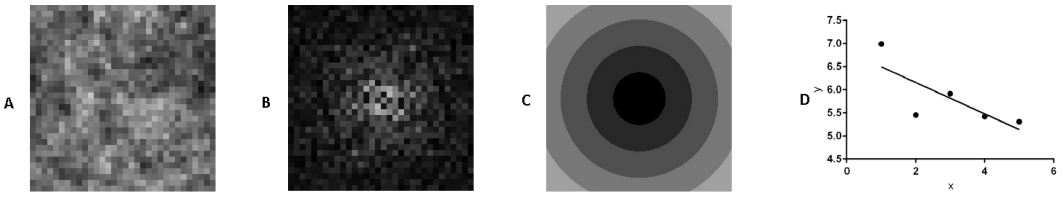
Tabuľka 2.1: Všetky testi boli prevádzané na 1mpx obrázku. V testoch sa počítala skutočná ostrosť pre 68 percent blokov. Hodnoty sú uvedené v milisekundách.

	Optm	Bez Optm
Výpočet 3m hodnôt	16	242
S1 algoritmus	791	1043
Beh celého programu	840	1102

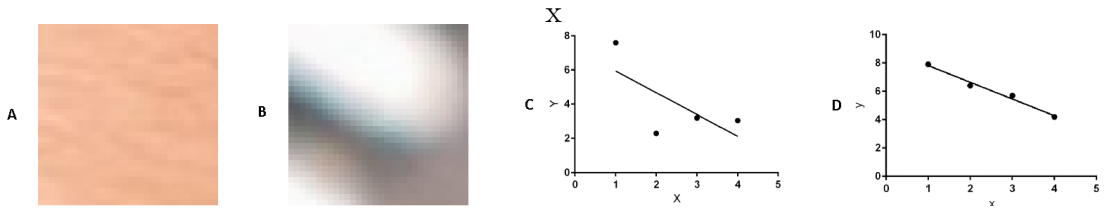
- 9 pre výpočet 3m v oblastiach *a1*, *a2*, *a3* a *a4*
- 9 pre výpočet 3m v oblastiach *b1*, *b2*, *b3* a *b4*

Uvedený postup je teda približne 16-krát rýchlepší čo sa aj ukázalo v praxi.
Pamäťová zložitosť. Výpočet blokov prechádza zhora nadol, zľava doprava po riadkoch. V každom momente si musíme pamätať navyše 3m informácie o častiach *a1*, *a2*, *a3* a *a4* pre jeden blok. Avšak 3m informácie o častiach *b1*, *b2*, *b3* a *b4* si pamätáme pre celý riadok nad práve počítaným riadkom. 3m informácia zahŕňa údaje o maxime, minime a priemere pre každú zo štyroch zón. Rádovo sa pamäťová náročnosť zvýšila o šírku obrázka. Čo je vzhľadom na to, že si musíme v pamäti udržiavať celý obrázok zanedbateľné.

Počítanie ostrosti Na implementáciu fourierovej transformácie bol použitý FFT algoritmus v čo najrýchlejšej možnej implementácii [5]. Napriek tomu, že FFT sa nepočíta pre bloky, ktoré neprešli filtrovaním jedná sa o výkonnostne kritický bod kvôli svojej nelineárnej časovej zložitosti. Aj keď počítanie FFT pre blok 32x32 nieje v čase $\mathcal{O}(n)$ ale $\mathcal{O}(n \log(n))$ celkovú časovú zložitosť nám to nezhorší kedže n je konštanta($n = 1024$). Lineárna regresia, ktorá odhaduje klesanie fourierovej transformácie je implementovaná pomocou metódy najmenších štvorcov. Vďaka malému počtu bodov, cez ktoré prekladáme priamku a dvojrozmernému priestoru je čas strávený nachádzaním priamky malý.



Obr. 2.4: **A)** Vstupný blok 32x32. **B)** 2D Fourierova transformácia vstupného bloku. **C)** Predpočítaná maska. Vyznačuje zóny, z ktorých sa počíta priemerná hodnota vo fourierovej transformácii. Priemerná hodnota pre každú zónu nám určuje jeden bod. **D)** Lineárnou regresiou odhadneme klesanie hodnôt jednotlivých bodov.



Obr. 2.5: **A)** Blok 32x32 pixelov zaostrenej pokožky. **B)** Blok 32x32 pixelov nezaostreného fotoaparátu.**C)** Klesanie fourierovej transformácie pre blok *A*. **D)** Klesanie fourierovej transformácie pre blok *B*.

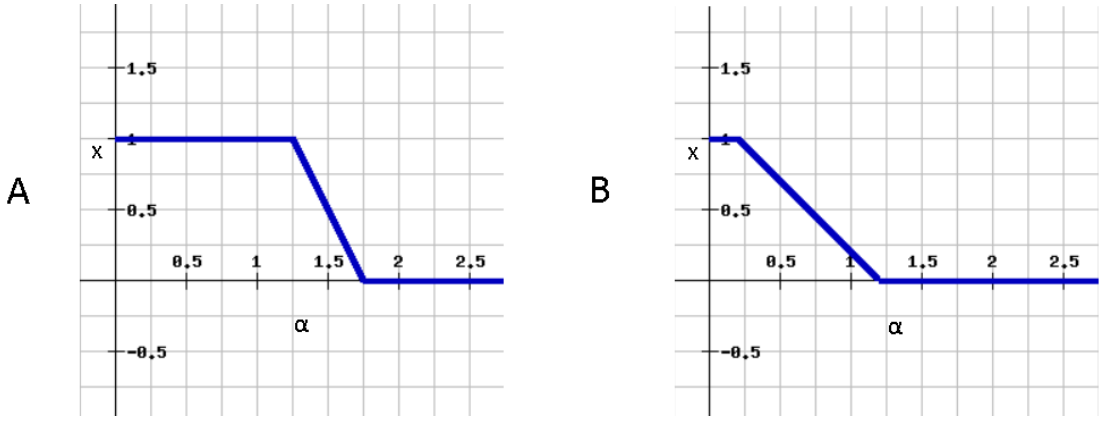
Problémy

Nasledujúca časť poukáže na nedostatky používaného algoritmu a spôsob ako sa týmito principiálne neriešiteľnými problémami vysporiadať. Obrázok 7., poukazuje na najzávažnejší problém. Ten spočíva v odlišných vlastnostiach hladkých a členitých povrchov. Ľudské oko vyhodnotí časť **A** z oblasti tváre ako zaostrenú, zatiaľ čo oblasť **B** ako nezaostrenú. Šum v bloku **A** je však viditeľne nižší ako šum v bloku **B** a tak aj odhad klesania Fourierovej transformácie týchto blokov nebude korešpondovať s realitou.

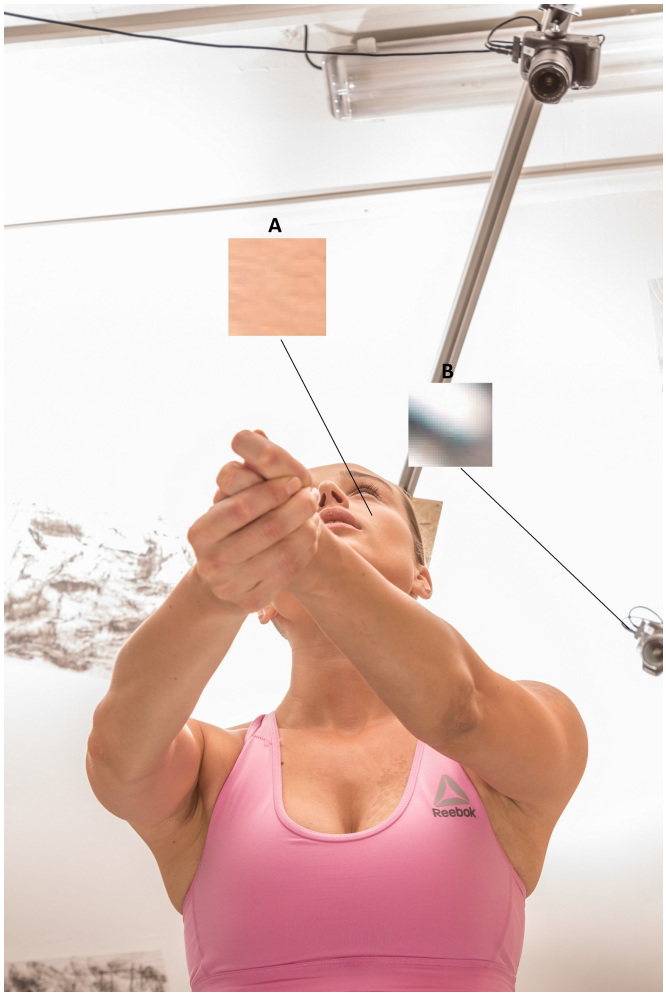
Hladké povrchy budú častokrát mylne považované za neostré a naopak členité povrchy budú mylne považované za ostré. Ľudský mozog pri rozpoznávaní obrazu rozumie jednotlivým predmetom a ich vlastnostiam. Na základe týchto vlastností posudzuje ostrosť povrchu. Bloky v oblasti fotoaparátu v obrázku 7. majú vysoký šum a prechádzajú nimi pomerne ostré hrany. Ľudský mozog vyhodnotí, že sa jedná o fotoaparát teda o objekt s veľmi členitým povrchom. Tým chápe, že nakriek tomu, že bloky z tejto oblasti majú pomerne vysoky šum, je nižší akoby pre dobre zaostrený fotoaparát mal byť. Demonštrovaný algoritmus avšak nerozpoznáva objekty v obraze, ku ktorým by následne pristupoval podľa ich vlastností. Namiesto toho používa rovnaký prístup k celému obrázku.
Poučenie. Ak chceme, aby výsledok ostrosti mapy *S1* korešpondoval s realitou je potrebné mať na vstupe objekty s podobnou vlastnosťou povrchu. Po tomto porovaní bol pridaný vstupný parameter programu, ktorý indikuje aký typ povrchu program može na vstupe očakávať.

- **true** indikuje vstup s hladkými povrchmi : $x = f_1(\alpha)$
- **false** indikuje vstup s členitými povrchmi : $x = f_2(\alpha)$

kde **x** je výsledná ostrosť bloku a α je uhol pod ktorým klesá funkcia odhadnutá lineárnou regresiou.



Obr. 2.6: **A)** Funkcia f_1 pre hladké povrchy. **B)** Funkcia f_2 pre členité povrchy.



Obr. 2.7: **A)** Blok 32x32 pixelov zaostrenej pokožky. **B)** Blok 32x32 pixelov nezaostreného fotoaparátu.

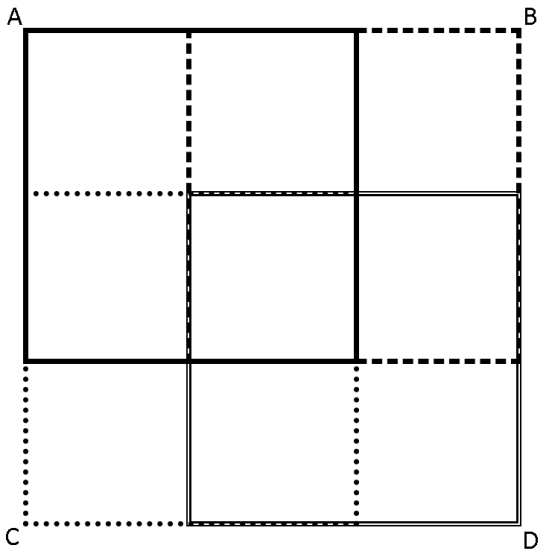
2.1.3 S2

Cieľom mapy *S2* je zachytiť ostré hrany. Je obecný predpoklad, že ostré hrany indikujú zaostrený obraz zatiaľ čo absencia ostrých hrán indikuje nezaostrený obraz.

Algoritmus

Nasleduje časť, ktorá stručne popisuje algoritmus na detekciu hrán. Jeho detailný opis je popísaný v tomto článku [2].

Bloky Podobne ako pri algoritme na výpočet *S1* mapy pixely obrázka rozdelíme do blokov. Tentokrát veľkosti 8x8 s prekrytím štyri pixely medzi susednými blokmi. Cieľom *S2*algoritmu bude opäť každému bloku dopočítať jeho ostrosť. Finálna ostrosť pixelu bude aritmetický priemer ostrosti blokov, ktoré sa nad daným pixelom prekrývajú.



Obr. 2.8: Prekrývanie blokov *A*, *B*, *C*, *D*.

Ostrosť bloku Ostrosť bloku x označme ako S(x). Blok rozdelíme na podbloky 2x2. Ostrosť podbloku y označme s(y). Ostrosť bloku x vypočítame ako

$$S(x) = \frac{1}{4} \max_{y \in \xi}(s(y))$$

kde ξ značí všetky podbloky 2x2 bloku x.

Ostrosť podbloku y vypočítame nasledovne

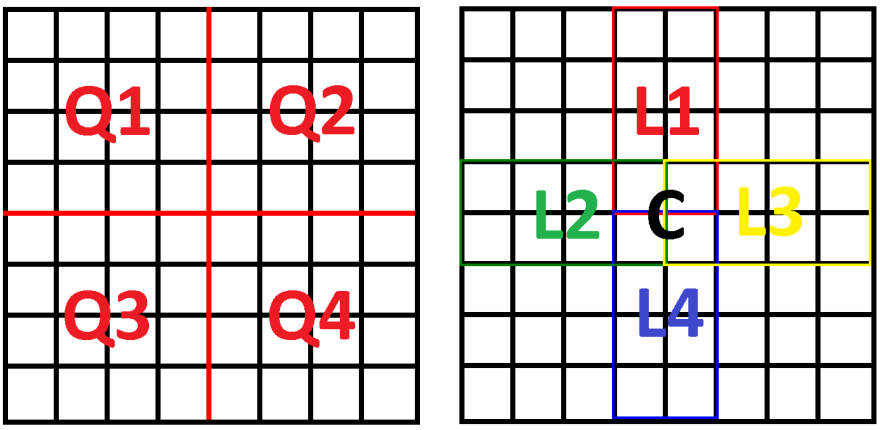
$$s(y) = \sum_{i,j} |x_i - x_j|$$

kde x_i a x_j sú hodnoty pixelov s indexami j,i v graysclale.

Implementácia Implentáciu daného algoritmu je možné spraviť priamočiaro. Ostrosť každého bloku spočítať zvlásť. Je vidieť, že takýto prístup nebude najefektívnejší. Oproti optimálnemu riešeniu, ktoré nebude počítať nič dva krát, bude pomalšie približne štvornásobne, kedže nad každým pixelom sa prekrývajú obecne štyri bloky. Použitá implementácia zvyšuje časovú efektivitu procesu za cenu zanedbateľného zvýšenia pamäťovej náročnosti.

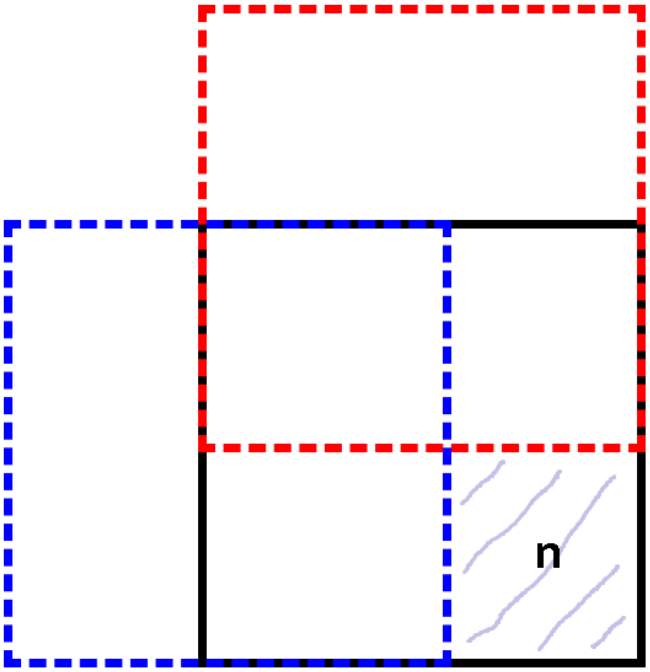
Použitá implementácia. Cieľom k vypočítaniu ostrosti bloku je násť podblok 2x2 s maximálnou ostrosťou. Preto si podbloky v bloku rozdelíme do deviatich častí $Q_1, Q_2, Q_3, Q_4, L_1, L_2, L_3, L_4, C$. Hodnota časti predstavuje najostrejší podblok 2x2 v nej obsiahnutý. Ostrosť bloku x spočítame nasledovne

$$S(x) = \frac{1}{4} max(Q_1, Q_2, Q_3, Q_4, L_1, L_2, L_3, L_4, C)$$



Obr. 2.9: Časti Q_1, Q_2, Q_3, Q_4 obsahujú deväť podblokov 2x2. Časti L_1, L_2, L_3, L_4 obsahujú tri podbloky 2x2. Časť C obsahuje len jeden podblok 2x2.

Podobne ako v predchádzajúcej časti predpokladajme, že predchádzajúce bloky práve počítanému bloku majú všetky tieto časti ($Q_1 ... C$) spočítané. Teda pre každú z týchto ôsmych častí vieme hodnotu najostrejšieho podbloku 2x2.



Obr. 2.10: Hodnotu naostrejšieho podbloku 2x2 pre všetkých osem častí máme pre červený a modrý blok vypočítané. Cieľom je dopočítať týchto osem častí aj pre čierny blok. Z maxima týchto ôsmych hodnôt dopočítame finálnu ostrosť bloku.

Štvorce budeme odteraz označovať nasledovne

- Čierny štvorec ako **C** (Current)
- Modrý štvorec ako **L** (Left)
- Červený štvorec ako **U** (Up)

Pre tieto štvorce platia nadchádzajúce závislosti.

- $C_{Q_1} = U_{Q_3}$
- $C_{Q_2} = U_{Q_4}$
- $C_{Q_3} = L_{Q_4}$
- $C_{L_1} = U_{L_3}$
- $C_{L_2} = L_{L_4}$

Na vypočítanie $S(C)$ stačí dopočítať C_{Q_4},C_{L_3},C_{L_4},C_C. Z celkových 41 $(4*9+4*3+1)$ podblokov 2x2 tým pádom treba spočítať 16 $(1*9+2*3+1)$.

Tabuľka 2.2: Všetky testi boli prevádzané na 1mpx obrázku. Hodnoty sú uvedené v milisekundách.

	Čas
S1 algoritmus	791
S2 algoritmus (optm)	52
S2 algoritmus (bez optm)	107

Záverečné porovnanie Čas strávený počítaním mapy *S2* je zanedbateľný oproti času strávenom na počítaní mapy *S1*. Tým pádom aj teoretické 2,5 krát zrýchlenie v konečnom dôsledku nehrá rolu. Optimalizácia sa stáva užitočnou v prípade generovania samotnej *S2* mapy.
Pamäťová zložitosť. Bloky opäť počítame zhora dole, zľava dobrava po riadkoch. Informácie o najostrejšom podbloku 2x2 pre časti $Q_1, Q_2, Q_3, Q_4, L_1, L_2, L_3, L_4, C$ si v jednom okamihu pamätáme pre všetky bloky v riadku nad práve počítaným riadkom. V stĺpci si stačí pamätať len jeden blok pred práve počítaným stĺpcom. Pamäťová náročnosť sa tým zvýšila o rádovo (šírka obrázka)bytov.

2.1.4 S3

Mapa *S3* je výsledkom celého procesu rozpoznávania ostrosti v obraze. Kombinuje techniky detekcie zrnitosti *S1* a detekcie hrán *S2*. Proces výpočtu mapy *S3* postupuje nasledovne.

1. Zmergovanie mapy *S1* a mapy *S2* (podla článku [2])
2. Finálne úpravy *S3* mapy (vlastná heuristika)

Mergovanie

Základ mapy *S3* dostane zmergovaním *S1* a *S2* máp. Mergovanie po pixeloch avšak nie je najefektívnejšie.
Rozlíšenie mapy *S1* je (x/8,y/8) a mapy *S2* je (x/4,y/4) (x - šírka vstupného obrázka, y - výška vstupného obrázka). Megovanie po blokoch 4x4 docieli rovnakého výsledku ako mergovanie po pixeloch. Rozlíšenie *S3* mapy bude teda 16-krát nižšie ako rozlíšenie vstupného obrázku.

$$S_3(x) = S_1(x)^{1/2} * S_2(x)^{1/2}$$

Finálne úpravy

Záverečné úpravy *S3* mapy sú motivované získaním lepšej pravdepodobnostnej mapy pre binárne segmentovanie na základe ostrosti. Vychádzajú z heuristiky, podľa ktorej je zaostrený objekt v strede obrazu obklopený neostrými oblasťami.

Tento objekt môže prípadne predchádať cez spodný okraj obrázku.
Pozorovanie. Častým prípadom je, že dobre zaostrený objekt obsahuje kdesi vo vnútri hladké časti. Tieto hladké časti budú avšak klasickým postupom vyhodnotené ako neostré. V binárnej segmentácii, pre ktorú nám *S3* poslúži ako pravdepodobnostná mapa budeme predpokladať, že zaostrený objekt neobsahuje "diery".

Úpravou *S3* mapy sa budeme snažiť časti, ktoré sú obklopené ostrosťou, ale sami boli vyhodnotené ako neostré detekovať a ich ostrosť umelo zvýšiť.

Počítanie vzdialenosti Pre každý pixel mapy *S3* spočítame najkratšiu cestu k okraju mapy. Cesty, ktoré vedú k spodnému okraju nebudeme rátať, kedže z heuristiky predpokladáme, že objekt môže prechádzať cez spodný okraj. Najkratšie cesty sa počítajú pomocou geodezickej vzdialenosti, ktorá sa popisuje v časti 2.5.2. V tomto konkrétnom prípade upravíme cenu za prechod medzi pixelmi

$$d(x,y) = max(0, (S_3(y) - S_3(x)) * S_3(y))^2$$

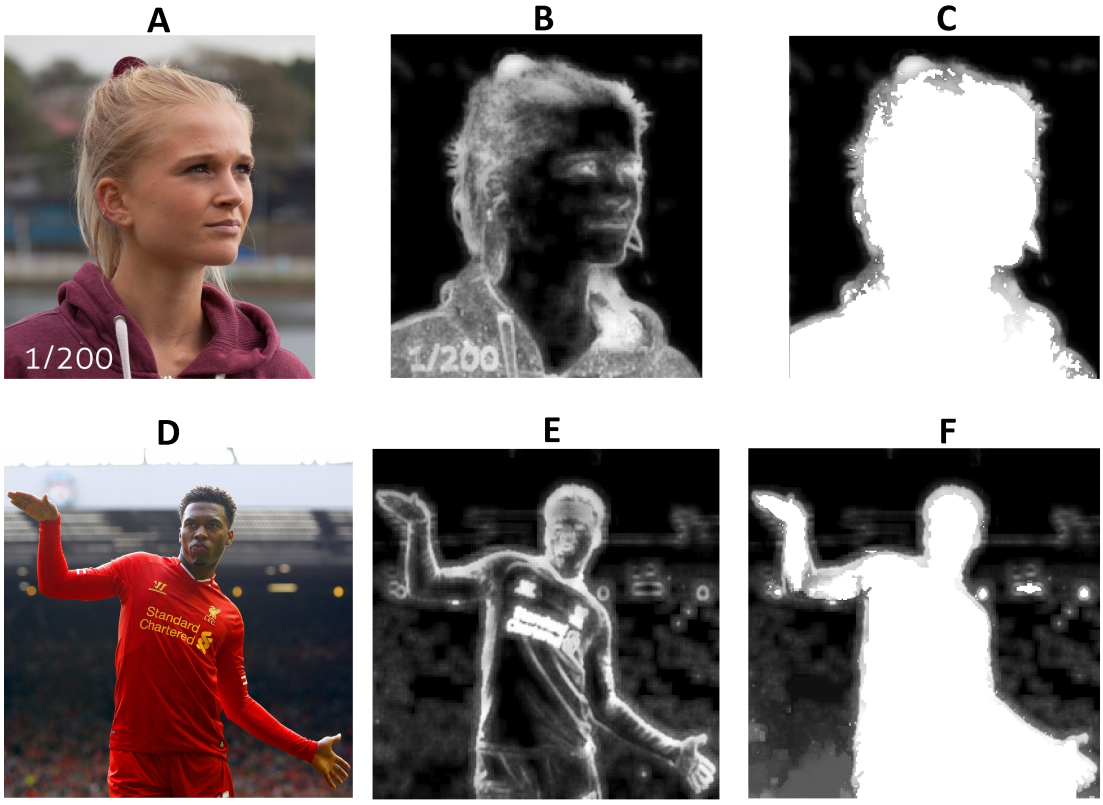
Cenu za prechod platíme len v prípade, ak prechádzame z menej ostrého pixelu na viac ostrý pixel. Za prechod zaplatíme viac v prípade, že celková ostrosť pixelov je vyššia. Vzdialenosť pixelu x je nakoniec

$$D(x) = \min_{y \in \xi}(d(x,y))$$

kde ξ sú pixele ľavého, pravého a horného okraja obrázku. Finálna úprava ostrosti každého pixelu sa upraví nasledovne

$$p(x) = max(S_3(x), (D(x) - t_1)/t_2)$$

Konštanty t_1 a t_2 boli nastavené na hodnoty 80000 a 150000.



Obr. 2.11: **A,D** Vstupné obrázoky. **B,E** *S3* mapy obrázkov A a D. **C,F** Úprava *S3* máp obrázkov A,D na základe vzdialenosti k okrajom.

2.2 Interaktívna segmentácia

Interaktívna segmentácia má za úlohu vylepšiť pravdepodobnostnú mapu automatickej segmentácie. Užívateľ myšou vyznačí dve množiny pixelov. Jednu k popredíu, druhú k pozadiu. Každý vyznačený pixel nesie v sebe dve relevantné informácie, ktoré nám môžu pomôcť čo najpresnejšie namodelovať popredie/pozadie. Farbu a pozíciu v obraze. Vytvoríme dva modely. Jeden založený na farbe, druhý na polohe. Modely budú vytrénované na interktívne vyznačených pixeloch. Následne každý pixel z obrazu bude v týchto modeloch ohodnotený a na základe hodnotenia mu bude pridelená výsledná pravdepodobnosť.

2.2.1 Farba

Prvý krok bude na základe vyznačených pixelov vytvoriť dva farebné modely (pozadie/popredie). Ako trénovacie dáta pre každý z týchto modelov budú použité farby z vyznačených pixelov popredia/pozadia. V druhom kroku bude farba každého pixelu z obrazu v týchto dvoch modeloch ohodnotená. Na základe pomeru ohodnotení bude vypočítaná finálna hodnota. Ďalej si musíme rozmyslieť dve dôležité časti: 1) výber vhodného modelu 2) reprezentácia farby.

Model

Ako vhodný model sme si vybrali zmes normálnych rozdelení(Gaussian mixture model) [6](pre popredie aj pozadie vytvoríme vlastný model). Ďalej si treba vhodne zvoliť počet komponent zmesi normálnych rozdeleni. Čím vyšší si ich počet zvolíme, tým presnejší model dostaneme. Nevýhodou príliš vysokého počtu týchto komponent je dlhšia doba modelovania. Pre náš konkrétny prípad sme sa rozhodli použiť dve komponenty zmesi normálnych rozdeleni, kedže predpokladáme, že farebná diverzita vyznačených pixelov nebude príliš veľká. Vytrénovanie modelu, vďaka malému počtu trénovacích dát nepatrí medzi výpočtovo kritické sekcie. Výpočtovo náročnejšie je ohodnotenie každého pixelu obrazu vo vytvorených modeloch, kedže sa počíta pre omnoho väčší počet pixelov ako bol počet trénovacích dát. Po ohodnotení každého pixelu v dvoch farebných modeloch je jeho finálna pravdepodobnosť pomocou sigmoidovej funkcie [7] spočítaná nasledovne

$$pd(x) = \log_{10}(p_1(x)) - \log_{10}(p_2(x))$$

$$p(x) = 1/(1 + \exp(-pd(x)/z))$$

$p_1(x)$ je ohodnotenie pixelu x vo farebnom modeli vytvorenom z trénovacích dát z popredia($p_2(x)$ z pozadia). Konštanta z nám určuje distribúciu funkcie

$p()$(vo všetkých testoch $z = 10$). Kedže ohodnotenia pixelov v modeli boli príliš nízke(rádovo nižšie než 10^{-10}), použili sme funkciu $\log_{10}$ ktorá nám hodnoty znormovala. Sigmoidová funkcia bola vybraná z dôvodu naškálovania výsledkov na interval [0,1], kde ak

$p(x) > 1 - \varphi \Leftrightarrow$ farba pixelu x je podobná farbe popredia(vzdialená pozadiu)

$p(x) < \varphi \Leftrightarrow$ farba pixelu x je podobná farbe pozadia(vzdialená popredíu)

φ je malé kladné reálne čislo. Počítať funkciu exp je pre veľké obrázky výpočtovo príliš náročné. Preto bol použitý výpočtovo jednoduchší odhad tejto funkcie [8].

$$exp(x) \doteq lim_{n->\infty}(1 + \frac{x}{n})^n$$

za n bola dosadená hodnota 1024.

Reprezentácia farby

Existuje mnoho rôznych reprezentácií farby a pre každú dostaneme iné výsledky. Časovo najoptimálnejšie bude použiť RGB reprezentáciu, kedže vstupné hodnoty očakávame práve v tomto formáte a nebudeme musieť robiť žiadne prevody na inú reprezentáciu. Vzdialenosť dvoch farieb v priestore reprezentujúcom farbu by mala byť v ideálnom prípade ekvivalentná k odlišnosti týchto farieb pri vnímaní ľudským okom.

$$D_{Space}(x, y) \Leftrightarrow \theta$$

$D(x, y)$ je euklidovská vzdialenosť medzi bodmi x, y. Space je farbený priestor, x, y sú farby, θ je rozdielnosť farieb x, y vnímaná ľudským okom. Farebná reprezentácia CIE LAB spĺňa uvedenú požiadavku lepšie ako RGB. Z tohto dôvodu bola pridaná možnosť modelovať aj nad CIE LAB farebným priestorom. Možnosť modelovať nad RGB bola zachovaná predovšetkým kvôli svojej najnižšej časovej náročnosti.

2.2.2 Vzdialenosť

Interaktívne označenie pixelu x ako popredia/pozadia nám napovedá, že tento pixel naozaj patrí do popredia/pozadia. Čo je však dôležité, takisto nám to čosi napovedá o pixeloch blízkych vyznačeným pixelom. Práve tento jav sa snaží zachytiť modelovanie pomocou vzdialenosti. Na počítanie vzdialenosti budeme opať používať geodetickú vzdialenosť popísanú v časti 2.5.2. Pre každý pixel obrazu budeme počítať dve vzdialenosti. Jednu k najbližšiemu vyznačenému pixelu z pozadia, druhú z popredia. Finálna vzdialenosť sa nakoniec vypočíta ako

$$d(x) = 1 - (MaxD + \min_{y \in \xi}(d(x,y)) - \min_{z \in \zeta}(d(x,z)))/(2 * MaxD)$$

$MaxD$ je konštanta určujúca najväčšiu možnú vzdialenosť od popredia/pozadia ($MaxD = 700$ vo všetkých testoch). $d(x,y)$ je geodetická vzdialenosť pixelov x,y. Množina ξ obsahuje všetky vyznačené pixely z popredia(ζ z pozadia). Použitím uvedeného vzorca sa nám vzdialenosti všetkých pixelov $d(x)$ znormujú do intervalu [0,1] kde platí

$$d(x) = 1 \Leftrightarrow \text{x je najbližšie popredíu (najďalej pozadiu)}$$

$$d(x) = 0 \Leftrightarrow \text{x je najbližšie pozadiu (najďalej popredíu)}$$

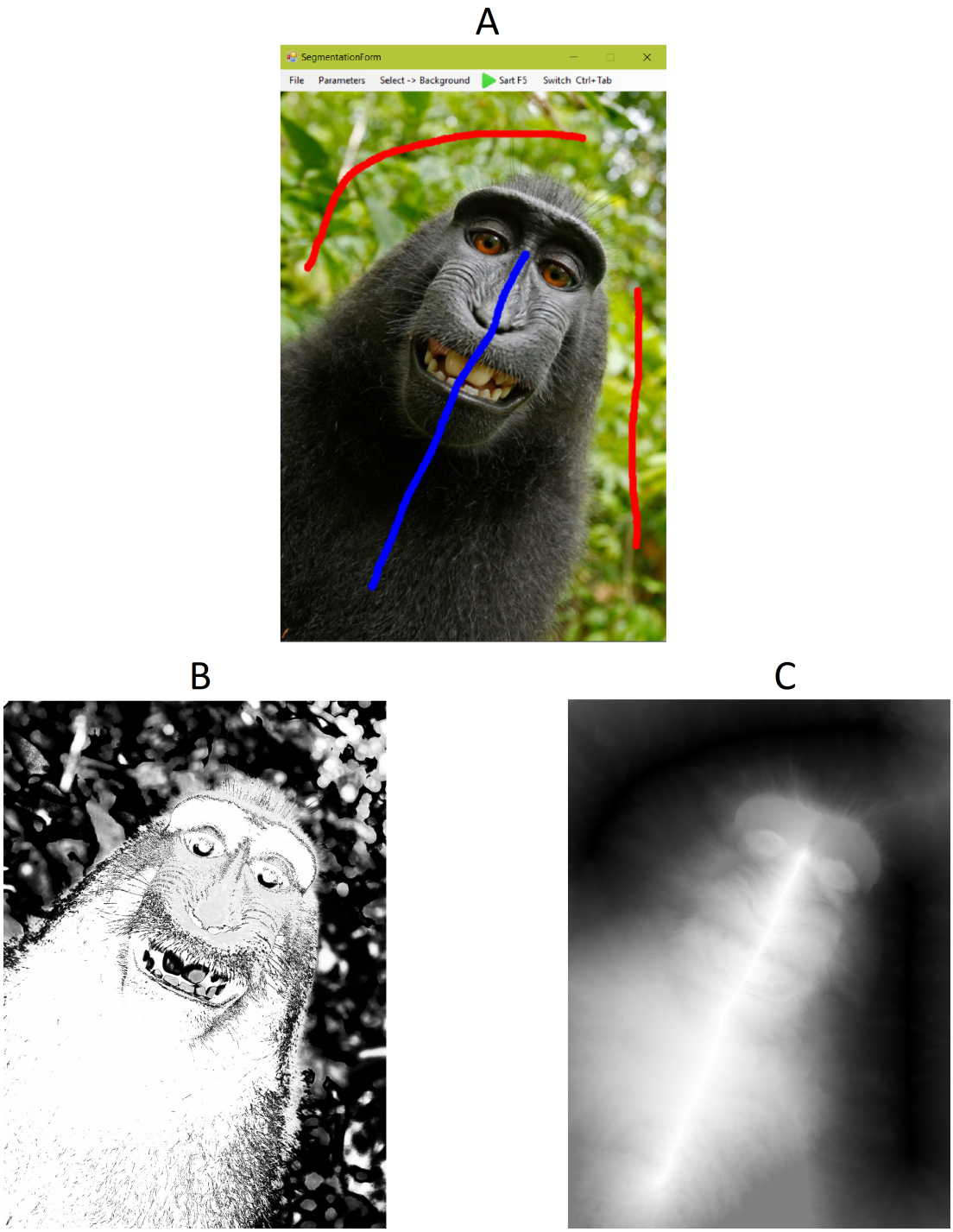
2.2.3 Zlievanie

V závere je potrebné ohodnotenia pixelov v jednotlyvých modeloch zlúčiť do jednej finálnej pravdepodobnosti. Keďže interaktívna segmentácia pracuje len ako nadstavba nad automatickou segmentáciou, máme pre každý pixel tri hodnoty z internalu [0,1]

- Ohodnotenie vo farebných modeloch (f(x))
- Ohodnotenie podľa vzdialenosti (d(x))
- Pravdepodobnosť z automatickej segmentácie (a(x))

Po testovaní viacerých funkcii bola nakoniec zvolená funkcia aritmetického priemeru.

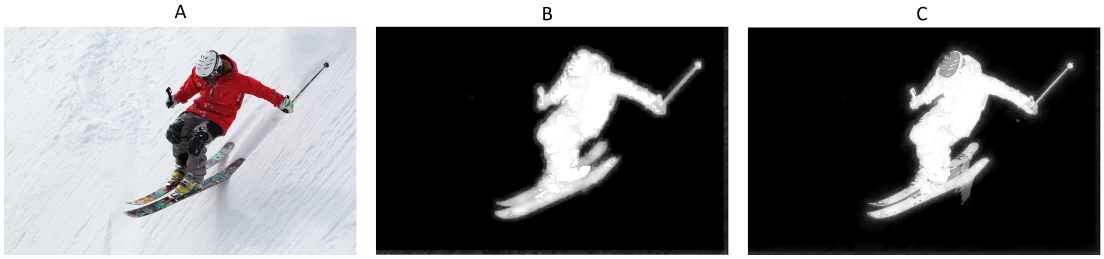
$$p(x) = (f(x) + d(x) + a(x))/3$$



Obr. 2.12: **A**) Vstupný obrázok s interaktívnou pomocou. **B**) Pravdepodobnosti z farebného modelu($p(x)$). **C**) Vzdialenosti $d(x)(\alpha = 0.5)$

2.3 S3 & Color

Myšlienka rozšíriť *S3* algoritmus o heuristiku na základe farby vznikla až počas vývoja. Je založená na nástrjoch používaných pri interaktivnej segmentácii. Postup je motivovaný pozorovaním, že pixely vyhodnotené *S3* algoritmom ako veľmi ostré, boli s vysokou pravdepodobnosťou z popredia(v groundtruth) a na-



Obr. 2.13: **A**) Vstupný obrázok. **B**) Pravdepodobnostná mapa *S3*. **C**) Pravdepodobnostná mapa S3 & Color.

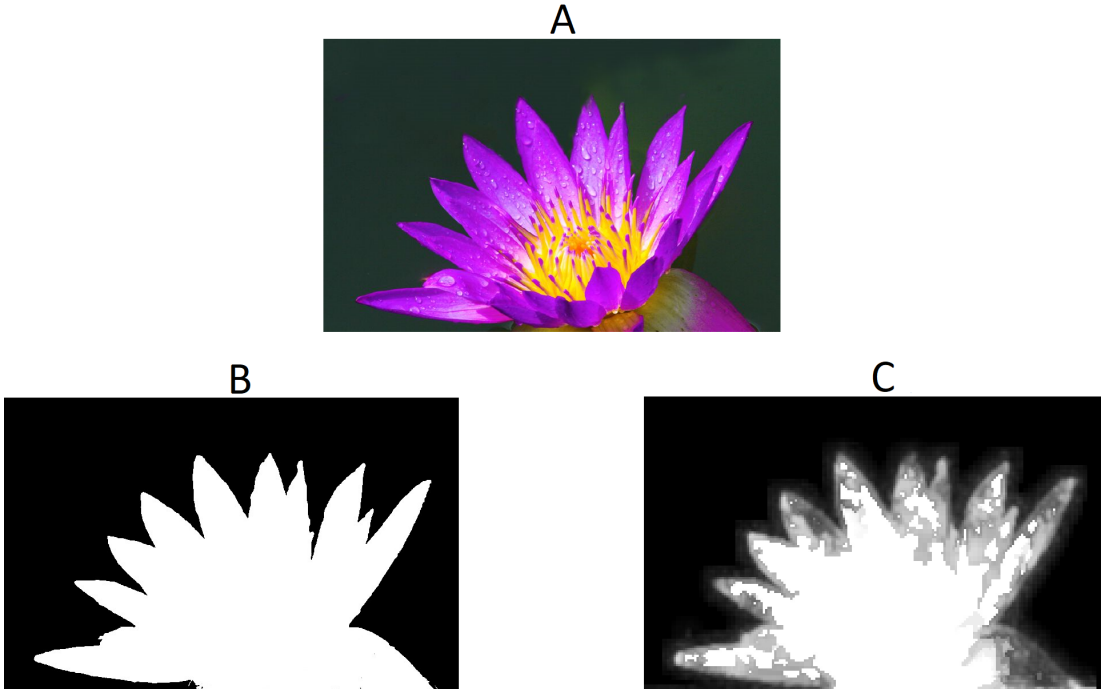
opak. Celý proces prebieha nasledovne

1. Výpočet pravdepodobnostnej mapy *S3*.
2. Určenie množín pixelov s veľmi vysokou/nízkou ostrosťou (konkrétne 0.95 a 0.05).
3. Týmto množinám priradíme rovnaký význam ako interaktívne vyznačeným pixelom
4. Finálna pravdepodobnostná mapa je vypočítaná podľa postupu v časti *Interaktívna segmentácia*

Proces výpočtu je veľmi podobný ako interaktívna segmentácia, až na fakt, že interaktívne vyznačené pixely nahradíme pixelmi ktoré boli *S3* algoritmom vyhodnotené ako veľmi ostré/neostré. Nevýhodou S3 & Color zostáva dlhší čas výpočtu, než samotný *S3* algritmus. Pri implementácii boli zvažované dva postupy ako vybrať množinu ostrých/neostrých pixelov.

1. Vybrať určité percento najostrejších/najneostrejších pixelov(napr. 1%).
2. Dopredu určiť max/min hranicu ostrosti.

Nakoniec bol implementovaný druhý spôsob, kvoli garancii ostrosti. V prípade, že sa v pravdepodobnostnej mape danou *S3* sa nenachádzajú dostatočne ostré/neostré pixely, nebude prezentované rozšírenie použité. Výsledky *S3* & *Color* sa ukázali byť oniečo zhodnejšie s groundtruth, než samotná *S3* mapa, avšak len v prípadoch, že popredie a pozadie bolo farebne odlišné. V opačnom prípade, mohly byť výsledky *S3* & *Color* dokonca horšie než *S3* mapa.



Obr. 2.14: **A**) Vstupný obrázok. **B**) Pravdepodobnostná mapa získaná farebnou segmentáciou **C**) Pravdepodobnostná mapa *S3*.

2.4 Farebná segmentácia

Automatická binárna segmentácia podľa farby je odbočka od našej hlavnej oblasti záujmu. Vznikla s úmyslom využiť naprogramované postupy k novému účelu. Výsledky sa však neukázali byť dostatočne uspokojujúce. Preto sa o segmentácii podľa farby zmienime len okrajovo.

2.4.1 Postup

Ako základ segmentácie podľa farby použijeme opäť zmes normálnych rozdelení(Gaussian mixture model). Tentokrát však použijeme len jeden model(opäť s využitím dvoch gaussianov). Ako trénovacie data použijeme všetky pixely segmentovaného obrazu. Ak je splnený náš predpoklad, že obraz je tvorený predovšetkým dvoma dominujúcimi farbami(a farbami im podobnými), mali by trénovacie data obsahovať dva dominujúce zhluky bodov vo farebnom priestore. Následne by každá z komponent zmesi normálnych rozdelení mala jeden z týchto zhlukov zachytávať. Výsledná pravdepodobnosť je vypočítaná ako pomer príslušnosti pixelu do vymodelovaných komponent. Konkrétna implementácia sa drží článku [6].

2.5 Geos

Cieľom *Geos* algoritmu je na základe pravdepodobnostnej mapy určiť finálnu binárnu segemntáciu. Samotný *Geos* už nijak nerozlišuje medzi spôsobmi akými bola pravdepodobnostná mapa nadobudnutá(S3, interaktívna segmentácia ...). Celý proces výpočtu je navrhnutý podľa článku [4] a nasledujúca časť popisuje jeho kľúčové časti nevyhnutné pre pochopenie fungovania.

2.5.1 Štruktúra algoritmu

Geos algoritmus si môžme rozdeliť na dve hlavné časti.

1. Výpočet symetrickej vzdialenosti
2. Minimalizovanie energie

Obe časti intenzívne využívajú *Geodetickú* vzdialenosť, preto si počítanie týchto vzdialeností vysvetlíme ako prvé.

2.5.2 Geodetická vzdialenosť

Počítanie geodetickej vzdialenosti je navrhnuté podľa článku [9]. Vďaka vlastnostiam dynamického programovania dosahuje lineárnu časovú zložitosť. Geodezická vzdialenosť počíta vzdialenosť v obraze. Narozdiel od obyčajnej vzdialenosti pridáva extra cenu cestám, ktoré prechádzajú cez hrany. Váhu, ktorú priraďujeme prechodom cez hrany parametrizujeme vstupným parametrom α. Geodezická vzdialenosť $D()$ pixelu x je definovaná ako

$$D(x) = \min_{y \in M}(d(x,y))$$

Množina M obsahuje všetky pixely obrazu a vzdialenosť $d(x, y)$ je dĺžka najkratšej cesty medzi pixelmi x,y.

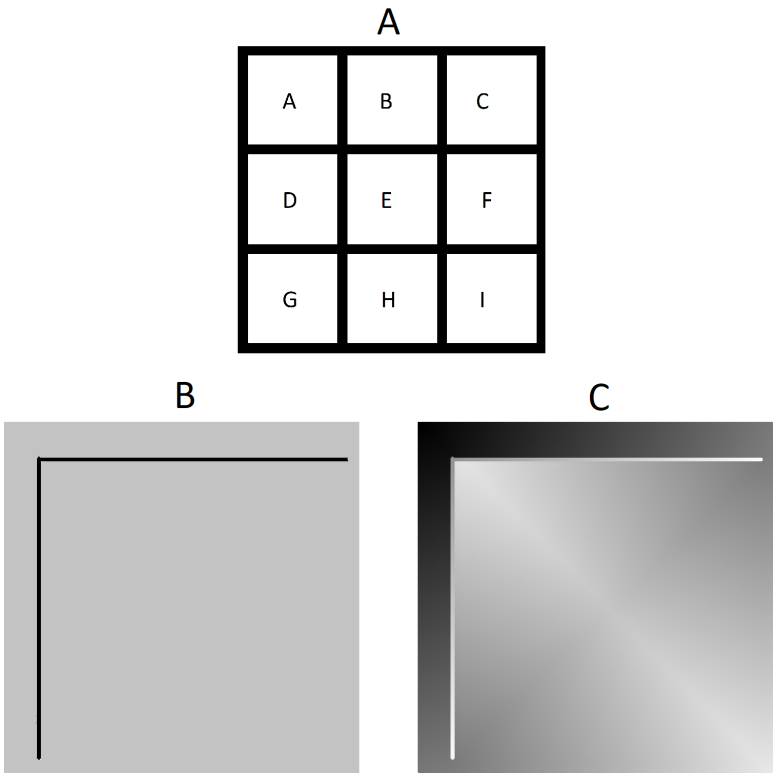
Celý proces výpočtu geodezickej vzdialenosti sa skladá z troch krokov.

1. Inicializácia vzdialenosti každého pixelu na jeho počiatočnú hodnotu(napr. $D(x) = 0, D(x) = \infty$).
2. Prechod každým pixelom zhora-dolu, zľava-doprava a úprava vzdialenosti

$$D(E) = \min(D(A) + d(A,E), D(B) + d(B, E), D(C)+ \\ d(C, E), D(D) + d(D, E), D(E)).$$

3. Prechod každým pixelom zdola-hore, zprava-doľava a úprava vzdialenosti

$$D(E) = \min(D(F) + d(F,E), D(G) + d(G, E), D(H)+ \\ d(H, E), D(I) + d(I, E), D(E)).$$



Obr. 2.15: **A**) Kernel 3x3 nad pixelmi obrazu. **B**) Vstupný obrázok s jednou ostrou hranou. Vzdialenosť pixelu 0,0(vľavo-hore) inicializovaná na nula(ostatné ∞). **C**) Geodezické vzdialenosti obrázku B. Čierna značí malú vzdialnosť, biela veľkú vzdialenosť.

Vzdialenoť susediacich pixelov x,y vypočítame ako

- $d(x,y) = 1 + \alpha \mid g(x) - g(y) \mid$ (x,y majú spoločnú stenu)
- $d(x,y) = \sqrt{2} + \alpha \mid g(x) - g(y) \mid$ (x,y nemajú spoločnú stenu)

$g(x)$ je hodnota pixelu x v šedotónovom obraze. Po vykonaní všetkých troch krokov nemusí byť vzdialenosť D() skutočne správna vzdialenosť pre všetky pixely. Jedná sa len o horný odhad. Opakovaním 2. a 3. kroku budeme dostávať presnejšie výsledky. Po dostatočnom opakovaní krokov 2. a 3. budeme mať garanciu presných výsledkov. Počet týchto opakovaní by však musel dosahovať veľké počty(rádovo max[šírka, výska obrazu]) čo by viedlo k ohromnému spomaleniu. Ako kompromis bol pri implementácii empiricky zvolený počet opakovaní 2. a 3. kroku na šesť.

2.5.3 Symetrická vzdialenosť

Symetrická vzdialenosť slúži k určeniu vzdialenosti pixelu od okraju objektu. Stavia na pravdepodobnostnej mape a geodezickej vzdialenosti. Symetrická vzdia-

lenosť pixelu x je definovaná podľa

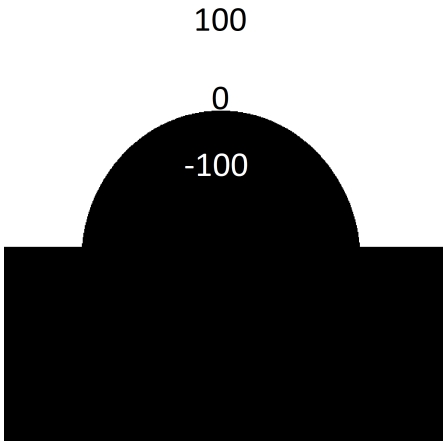
$$D_s(x) = D(x) - D_c(x)$$

kde $D_c(x)$ je geodezická vzdialenosť pixelu x od doplnku. Na výpočet symetrickej vzdialenosti je potreba spočítať dve geodezické vzdialenosti. Tie spočítame podľa postupu opísanom v predchádzajúcej časti. Líšiť sa budú jedine v prvom kroku(inicializácia vzdialenosti)

$$D(x) = v * p(x)$$

$$D_c(x) = v * (1 - p(x))$$

kde $p(x)$ je hodnota pixelu x z pravdepodobnostnej mapy, v je konštanta, ktorá určuje maximálnu možnú vzdialenosť od objektu. Vzdialenosť symetrickej mapy nám napovedá ako ďaleko sa pixel nachádza od okraja objektu. Proces výpočtu symetrických vzdialenosti nie je časovo kritická sekcia vďaka lineárnej časovej zložitosti a vykonávaniu iba jednoduchých operácii ako násobenie, sčítavanie ... Implementácia bola teda navrhnutá priamočiaro bez potreby časovej optimalizácie.



Obr. 2.16: Čierna oblasť pre objekt, biela pre doplnok.

2.5.4 Minimalizovanie energie

Minimalizácia energie je posledný krok k výslednej binárnej segmentácii. Je založená na symetrických vzdialenostiach a na operátoroch erózie a dilatácie. Proces minimalizácie energie prebieha nasledovne

1. Na symetrické vzdialenosti aplikujeme filtre erózie a dilatácie (Upravené vzdialenosti po erózii a dilatácii nazývame D_s^s).

2. Z upravených vzdialeností (D_s^s) odvodíme binárnu masku (M_s^s).

3. Pre binárnu masku spočítame celkovú energiu.

Tento proces opakujeme pre rôzne parametre erózie a dilatácie. Pre rôzne parametre dostaneme iné $D_s^s()$ a tým aj rôzne energie. Ako finálnu binárnu masku vyberieme tú, pre ktorú dostaneme najnižšiu energiu.

Masky Maska je matica $M_{m,n}$, ktorá obsahuje jedine hodnoty 0,1. Ďalej platí, že m = šírka obrazu a n = výška obrazu. Hodnota v matici i,j hovorí, či pixel $x_{i,j}$ patrí pozadiu alebo popredíu($1 \Leftrightarrow$ popredie). Masku M_s odvodíme zo symetrickej vzdialenosti D_s nasledovne

$$M_s(x) = [D_s(x) < 0]$$

kde [.] je operátor vracajúci 1 ak pravda(0 ak nepravda). Masku M_s^s dostaneme z D_s^s vzdialeností ako

$$M_s^s(x) = [D_s^s(x) < 0]$$

Erózia a dilatácia

Snaha upraviť pôvodné vzdialenosti D_s na nové D_s^s použitím operátorov erózie a dilatácie vyplýva z potreby odstrániť izolované časti(malé oblasti) v oblasti popredia a pozadia. Pretože aj neostré pozadie môže obsahovať ostrú hranu(kábel, stonka ...), a naopak ostré popredie môže obsahovať neostrú oblasť(pokožka ...). Operácia erózie odstraňuje malé oblasti vyhodnotené ako popredie z pozadia. Dilatácia naopak upravuje malé oblasti z popredia, vyhodnotené ako pozadie, na popredie. Intenzitu erózie a dilatácie určuje dvojica parametrov označovaných ako θ(θ_e pre eróziu, θ_d pre dilatáciu). Tieto parametre určujú ako veľké oblasti sme schopní odstrániť. Príliš vysoké parametre θ však môžu pokaziť pôvodné okraje objektu. Masku po aplikovaní erózie nazývame nazývame M_e a masku dilatácie M_d.

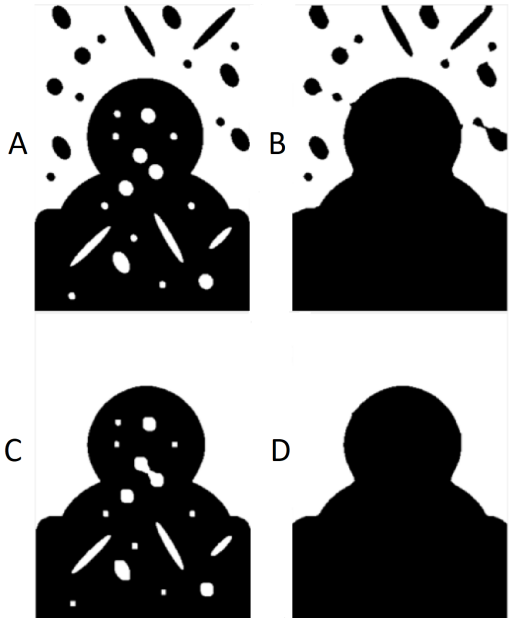
$$M_e(x) = [D_s(x) < \theta_e]$$

$$M_d(x) = [D_s(x) < -\theta_d]$$

Masky M_e a M_d využijeme na počítanie D_s^s vzdialeností, z ktorých odvodíme finálnu M_s^s masku zachytávajúcu operácie erózie a dilatácie.

$$D_s^s(x) = D_{M_e}(x) - D_{\overline{M_d}}(x) - \theta_e + \theta_d$$

kde D_{M_e} je geodezická vzdialenosť inicializovaná maskou M_e a $D_{\overline{M_d}}$ je geodetická vzdialenosť inicializovaná doplnkom masky M_d.



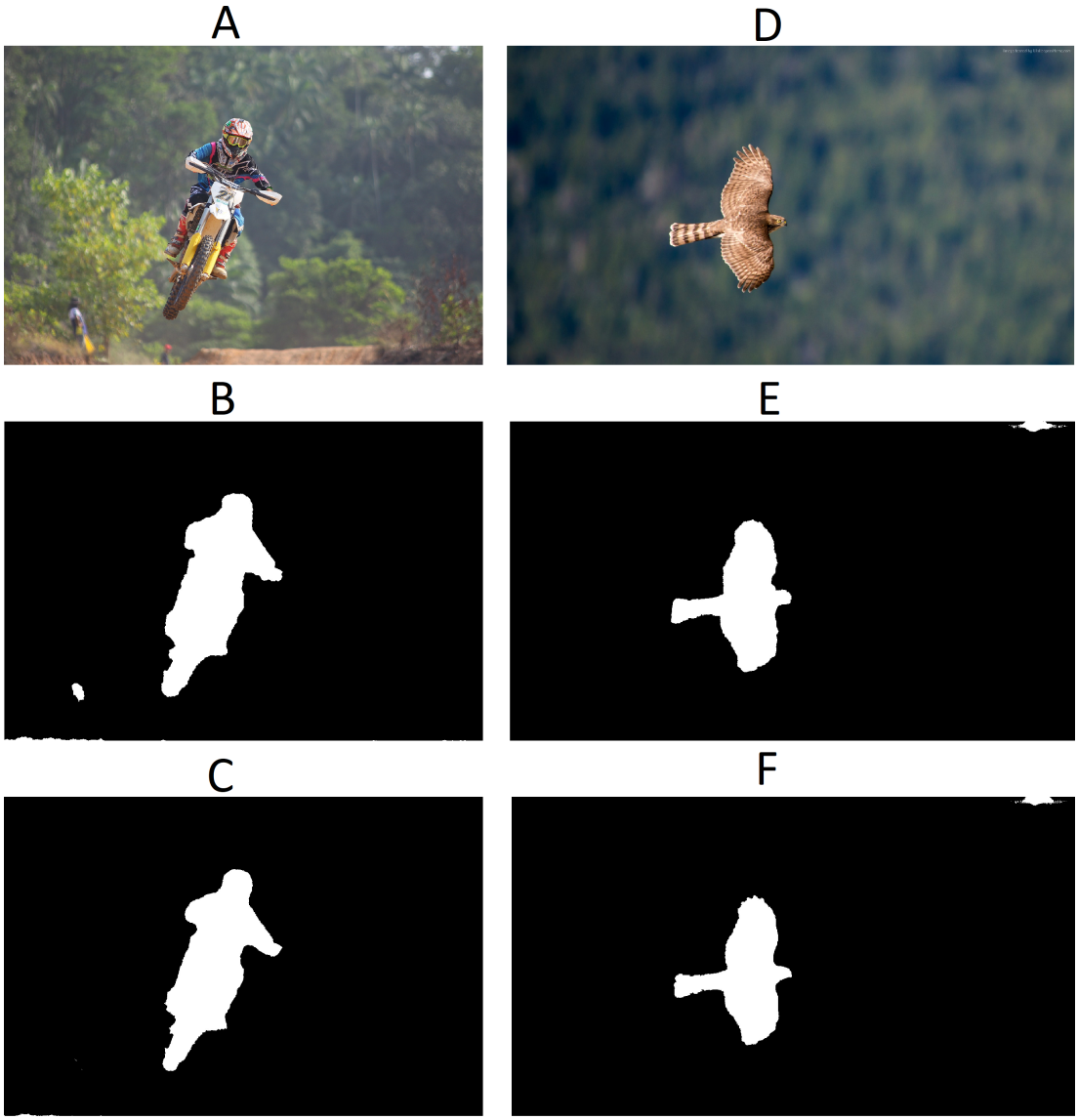
Obr. 2.17: **A**) Maska M_s získaná zo symetrických vzdialeností D_s(Čierna $\Leftrightarrow$ popredie, Biela $\Leftrightarrow$ pozadie). **B**) Pôvodná maskaM_s upravená operátorom dilatácie M_d. **C**) Maska M_e po erózii masky M_s. **D**) Finálna makska M_s^s odvodená z vzdialeností D_s^s.

Energia

Už vieme, ako pomocou erózie a dilatácie spočítame masku M_s^s. Pre rôzne parametre θ dostaneme iné M_s^s. Vyššie hodnoty θ vedú k odstráneniam rozľahlejších oblastí z popredia/pozadia, a tiež k hladším hraniciam medzi objektom a pozadím. Masku dosiahnutú po aplikovaní erózie a dilatácie s parametrami θ_i značíme $M_s^s[\theta_i]$. Pre finálnu segmentáciu, ktorú cheme dosiahnúť, platí

$$M_s^s[\theta_i], \theta_i = args\min_{\theta \in S}(E(M_s^s[\theta]))$$

S je množina všetkých možných θ parametrov. Detailný postup ako spočítať energiu masky $E(M)$ je popísaný v článku [4](rovnica 8.) Pre každý prvok z tejto množiny je potrebné spočítať **A**) masku $M_s^s[\theta_i]$, **B**) energiu $E(M_s^s[\theta])$. Najnáročnejší proces pri výpočte $M_s^s[\theta_i]$ masky je počítanie dvoch geodezických vzdialeností(M_e, $\overline{M_d}$). Výpočet energie masky $E(M_s^s[\theta])$ je avšak ešte náročnejší proces, kvôli zložitým operáciám(exp, log). Kedže je počítanie jednotlivých masiek a energií náročný proces, je potrebné precízne obmedziť mohutnosť množiny S. V našej implementácii sme sa rozhodli pre veľkosti dva, štyri a šesť. Vďaka jednoduchej logickej oddelitelnosti, sa jednotlivé masky a energie počítajú paralelne.



Obr. 2.18: **A, D**) Segmentované obrázky. **B**) Maska M_s obrázka A. **C**) Maska M_s^s obrázka A. **E**) Maska M_s obrázka D. **F**) Maska M_s^s obrázka D.

3. Programátorská dokumentácia

Táto kapitola je venovaná programátorskej dokumentácii. Program sa skladá z dvoch častí. **A**) Knižnica *Geos.Dll* vykonávajúca segmentáciu. **B**) Oknová aplikácia *SegmentationForm.exe*, ktorá zaisťuje interaktívne rozhranie, potrebné predovšetkým pri interaktívnej segmentácii. SegmentationForm sa dynamicky zlinkuje s Geos.Dll a tým vytvorí kompaktnú aplikáciu.

3.1 Konvencie

Pre lepšiu prehľadnosť kódu sú všetkých metódy anotované *SAL* anotáciami. Pomenovania atribútov tried sú prefixované prefixom "m_". Pokiaľ je premenná typu pointer alebo referencia je jej názov prefixovaný ako "p, r, pp(v prípade pointru na pointer)". Pre oznámenie neúspešnej akcie sa používa objekt *HRESULT*(najmä pri načítaní a ukladaní obrázkov z disku). Návratové hodnoty funkcii sú vrátené pomocou referencie cez posledný parameter funkcie. Zdrojvé súbory s príponou .h obsahujú definície funkcií, a súbory .cpp ich implementáciu. Zdrojové súbory súvisiace s výpočtom mapy pravdepodobností podľa ostrosti(S3) sú umiestnené v priečinku *SourceSharpness*. Všetky ostatné zdrojové súbory projektu *Geos.Dll* sú lokalizované v adresári *SourceGeos*.

3.2 Oknová aplikácia

Oknová aplikacia je napísaná v jazyku C# a vyvíjaná pomocou frameworku WindowsForms. Je spustiteľná na operačnom systéme Windows. Jej hlavným cieľom je preukázať funkčnosť knižnice "Geos.Dll". Ďalej zabezpečuje príjemnejšiu prácu so súborovým systémom pri načítaní a ukladaní fotografii, prehľadnejšie nastavovanie parametrov pre segmentáciu a možnosť interaktívnej segmentácie. Funkcionalita oknovej aplikácie je rozdelená na dve hlavné okná : *SegmentationForm* a *Editform*.

3.2.1 SegmentationFrom

Formulár *SegmentationForm* implementuje hlavnú funkcionalitu oknovej aplikácie. (načítanie fotiek, interaktívna pomoc, exekúcia segmentácie ...).

Načítanie fotiek je implementované pomocou *OpenFileDialog*. Samotná knižnica *Geos* vo svojej pôvodnej implementácii umožnovala časovo efektívne načítanie a ukladanie obrázkov z disku v .JPG alebo .TIF formáte do surovného poľa pomocou frameworku [10]. V spolupráci s oknovou aplikáciou túto úlohy priro-

dzene prebral *SegmentationForm*. Kvôli spätnej kompatibilite *SegmentationForm* zachováva množinu podporovaných formátov.

Exekúcia segmentácie volá metódu *SegmentationProcess* z knižnice *Geos.Dll*. V tejto metóde jej predáva všetky potrebné parametre(interaktívnu pomoc, cestu k vstupnému obrazu ...). Výsledná maska je vrátená referenčne v jednorozmernom poli, ktoré je takisto súčasť vstupných argumentov. Kvôli prevádzaniu parametrov medzi manažovaným(SegmentationFrom) a nemanažovaným(Geos) kódom, bolo potrebné zaistiť *marshalovanie*[11] parametrov metódy *SegmentationProcess*. Z tohto dôvodu boli argumenty metódy *SegmentationProcess* zvolené tak, aby podporovali implicitné *marshalovanie*. Kedže volanie funkcie segmentácie môže trvať netriviálne dlho, nechceme, aby po celú dobu výpočtu bola aplikácia zamrznutá, ale ideálne indikovala, že pracuje. Teda požadujeme

1. vykonať určitú akciu okamžite(oznámenie že program začal pracovať)
2. počas výpočtu neblokovať riadenie udalostí (kliknutie na tlačítko ...)
3. po skončení okamžite oznámiť výsledok
4. možnosť segmentáciu predčasne ukončiť

Pre vhodnejšie API a jednoduchšiu implementáciu sme sa rozhodli na implementáciu našich požiadaviek použiť triedu *Task*(Task.Factory) namiesto *Thread*. Možnosť predčasne zrušiť vykonávanú segmentáciu bola implementovaná pomocou *cancelationToken*.

3.2.2 EditForm

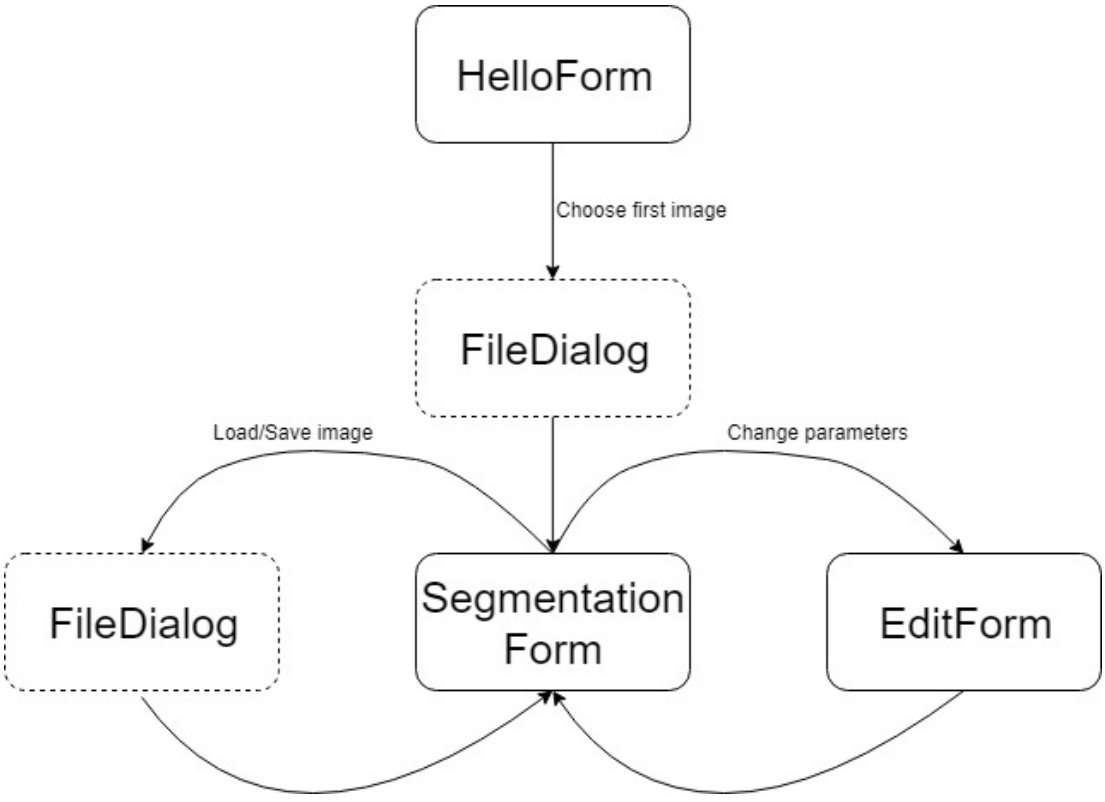
EditForm je veľmi jednoduchý formulár, ktorý slúži výhradne na zmenu parametrov pre segmenáciu. Všetky parametre sú uložené v objekte *SegmentationParams*.

3.3 Segmentácia

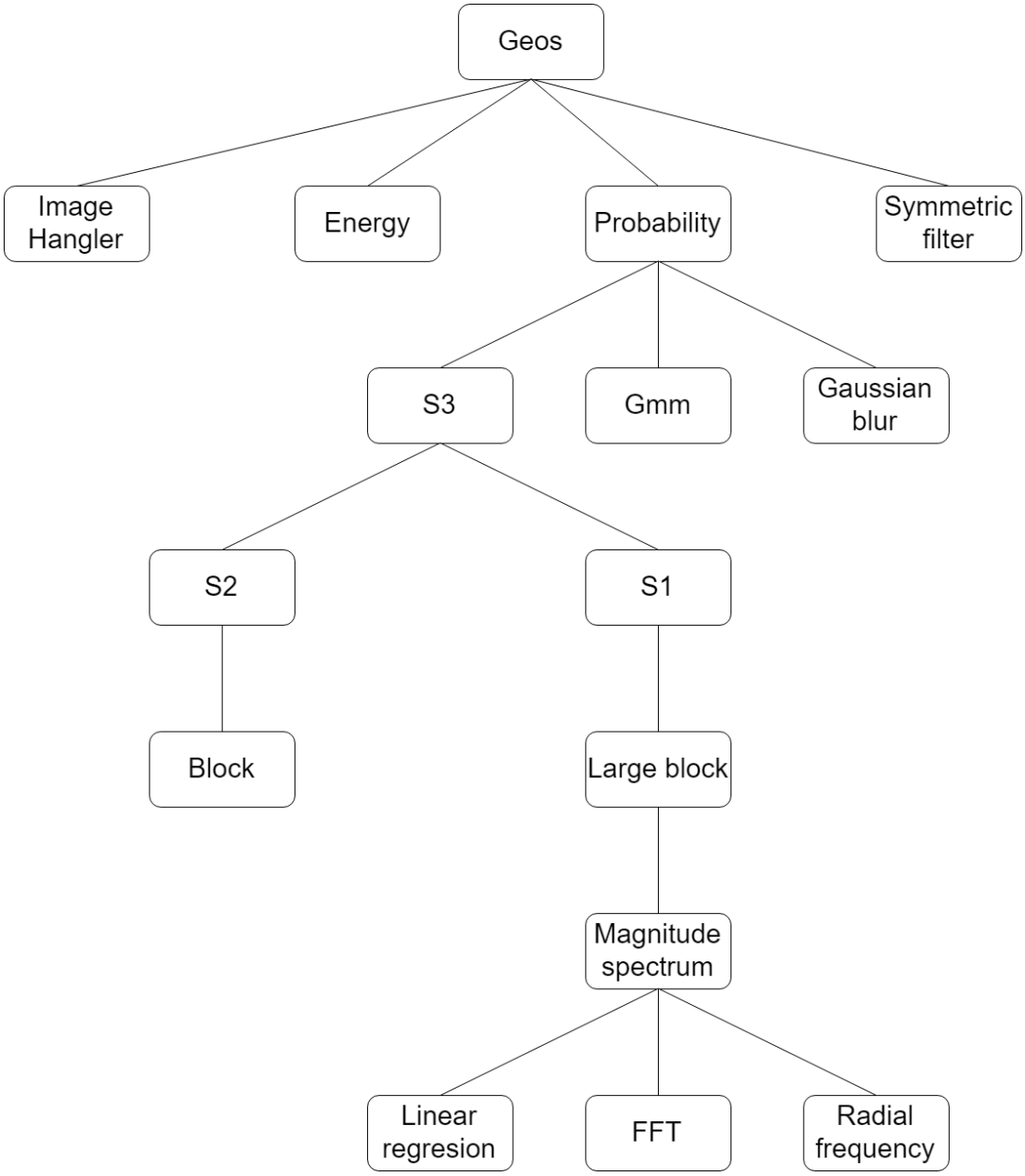
Nasledujúca časť popisuje implementačné riešenie binárnej segmentácie. Popisuje funkcionalitu jednotlivých objektov a aj to ako jednotlivé objekty medzi sebou súvisia. Štruktúra objektového návrhu tvorí strom v ktorom platí, že objekt k implementácii svojho API využíva API objektov jemu podriadených(S3 k implementácii mapy ostrosti využíva verejné API objektov S1 a S2). API koreňového objektu stromu(Geos) poskytuje finálnu segmentáciu obrazu.

Geos

Trieda *Geos* zastrešuje celý proces výpočtu segmentácie. Vstupným bodom je procedúra *Process*, ktorej sa predajú všetky potrebné parametre pre segmentáciu. Pôvodné API tejto metódy bolo upravené kvôli potrebe predávať všetky zvolené parametre pomocou *EditForm*. *Process* v prvom kroku načíta segmentovaný obrázok z disku pomocou *ImageHandler*. V druhom kroku zavolá metódu *ImageSegmentation*, ktorá zaisťuje samotnú segmentáciu. Výsledok segmentácie je uložený vo verejnej položke *m_ppResult*. Metóda *ImageSegmentation* si najskôr zistí pravdepodobnostnú mapu pomocou objektu *Probability*, ktorému predá informácie o tom aký typ mapy požaduje(S3, interaktívnu ...). Následne zavolá metódu *MinimalizeEnegry*, ktorá sa stará o výpočet finálnej masky. V nej počí-



Obr. 3.1: Štruktúra oknovej aplikácie.



Obr. 3.2: Štruktúra objektového návrhu. Každý objekt využíva funkcionalitu objektu jemu podriadeného.

tanie jednotlivých masiek a ich energií vykonáva objekt *Enegry*. Toto počítanie prebieha paralelne. Nakoniec sa vyberie maska s najnižšou enegriou.

Image Handler

Objekt *Imagehandler* zaisťuje načítanie a ukladanie obrázkov. V našom konkrétnom prípade využívame len načítanie, kedže o ukladanie sa stará okenná aplikácia. Pre čo najvyššiu efektivitu načítavania a práce s obrázkami sme použili framework popísaný v článku [10]. Pri vytváraní *ImageHandleru* špecifikujeme absolútnu cestu k vstupnému súboru. Pomocou metód *Create* a *Save* môžme vstupný súbor načítať a ukladať. Metóda *Create* vracia objekt *Image* ktorý obsahuje všetky potrebné dáta pre prácu s obrázkom(výšku, šírku, pole pixelov).

Energy

Objekt *Energy* sa stará o počítanie masky a jej energie. Pre každý parameter θ(erózie a dilatácie) je vytvorený jeden objekt tohto typu. Pri vytváraní objektu *Energy* sa konštruktoru predajú všetky potrebné informácie(pravdepodobnostná mapa, θ ...). Samotný proces sa spúšťa zavolaním metódy *EntryPoint*, ktorá **A**) spočíta novú masku, **B**) určí energiu novo spočítanej masky. K počítaniu novej masky využíva triedu *SymmetricFilter*. Kedže počítanie každej novej masky (nezávisle na parametre θ), sa odvíja od tých istých symetrických vzdialeností D_s, tieto vzdialenosti sú už predpočítané a predané objektu *Energy* v objekte *SymmetricFilter*. Počítanie energie a masiek prebieha podľa vzorcov uvedených v časti 2.5.4.

SymmetricFilter

SymmetricFilter sa stará o počítanie masiek pomocou geodezickej vzdialenosti, na základe pravdepodobnostnej mapy a parametrov θ(erózie a dlatácie). Symetrické vzdialenosti D_s sú odvoditelné už z pravdepodobnostnej mapy, preto sa tieto vzdialenosti počítajú len raz v konštruktore. Výsledné masky M_s^s sa počítajú v metóde *GetSymmetricMask* ktorej sa predajú už konkrétne parametre θ. Z tohto vyplýva, že je omnoho efektívnejšie objekt *SymmetricFilter* raz vytvoriť a následne na jednej instancii volať opakovane *GetSymmetricMask*(pre rôzne parametre θ), ako pre každý parameter θ vytvárať novú instanciu *SymmetricFilter*. Práve z tohto dôvodu sa pri vytváraní každej instancie *Energy* predá objekt *SymmetricFilter*.

Probability

Objekt *Probability* obsahuje jedinú verejnú metódu *GetProbability*, ktorá zaisťuje počítanie pravdepodobnostnej mapy pre každý typ segmentácie. V prvom

kroku vypočíta pravdepodobnostnú mapu pomocou automatickej segmentácie. V druhom(nepovinnom) kroku vylepší pravdepodobnostnú mapu pomocou interaktívnej segmentácie(ak je zvolená). Vlastné úpravy pravdepodobnostnej mapy *S3* vykonáva pomocou metód *ModifySharpness1*(vzdialenosť od okrajov viď. 2.1.4) a metódy *ModifySharpness2*(farebný model 2.3).

Gmm

Objekt *Gmm* poskytuje možnosť modelovať pixely podľa farby. Obsahuje dve verejné metódy. Automatickú segmentáciu podľa farby zabezpečuje metóda *AutomaticProbability*, ktorá implementuje postup opísaný v kapitole 2.4. Interaktívnu segmentáciu implementuje *InteractiveProbability*, ktorej sa predajú súradnice vyznačených pixelov. Tá narozdiel od *AutomaticProbability*, ktorá vytvára len jeden model, vytvára dva modely(jeden pre pozadie druhý pre popredie). Samotné trénovanie modelu vykonáva metóda *train* pomocou opakovaného striedania krokov *m_step* a *e_step*.

S3

S3 zabezpečuje počítanie pravdepodobnostnej mapy podľa ostrosti v podobe ako ju uvádza článok [2], bez modifikácii založených na vlastnej heuristike. Celý proces sa spúšťa metódou *GrayToResult*, ktorá z čierno-bieleho obrázku určí mapu ostrosti. Argumentmi tejto metódy sa dá parametrizovať výsledná mapa ostrosti(typ výslednej mapy[S1, S2, S3], členitosť povrchu[f_1, f_2,] viz. 2.1.2). Tieto parametre sa však používateľom nedajú presnejšie špecifikovať, a tak je metóda *GrayToResult* volaná vždy s tými istými parametrami(okrem čierno-bielej reprazentácie segmentovaného obrazu). Počítanie máp *S1* a *S2* prebieha paralelne.

S2

Trieda *S2* obsahuje jedinú verejnú metódu *CreateS2* ktorá počíta mapu ostrostí, na základe ostrých hrán, spôsobom opísaným v časti 2.1.3. Všetky potrebné informácie o bloku 8x8 sú obsiahnuté v triede *Blok*. Ostrosť prvého riadku blokov počíta metóda *CreateS2* priamočiaro(neefektívne) pomocou metódy *CountFirstRow*. Ostrosť všetkých ostatných blokov je už počítaná efektívne pomocou metódy *CountRow*, ktorá využíva informácie o ostrosti blokov predchádzajúcich. Hodnota ostrosti pixelu je určená aritmetickým priemerom ostrosti blokov nad týmto pixelom sa prekrývajúcich. Tento prepočet zaisťuje metóda *SetBuffer*.

Block

Trieda *Block*(8x8) reprezentuje ostrosť bloku 8x8 využívaného triedou*S2*. Obsahuje atribúty reprezentujúce ostrosť jednotlivých podčastí(viď 2.1.3), ako aj

metódy počítajúce ostrosť týchto častí(*CountLine*, *CountQuater*). Samotný *Block* nerozumie významu jednotlivých podčastí v kontexte viacerých blokov. Významu podčastí susediacich blokov(ktorých podčasti sa rovnajú) chápe trieda *S2*, ktorá túto znalosť využíva k efektívnemu počítaniu ostrosti jednotlivých podblokov.

S1

S1 podobne ako trieda *S2* obsahuje jedinú verejnú metódu *CreateS1*, ktorá implementuje výpočet mapy *S1*. Na určenie ostrosti jednotlivých blokov využíva triedu *LargeBlock*. Samotný výpočet ostrosti prebieha po riadkoch. Ostrosť prvého riadku počíta metóda *CountFirstRow* pomocou metód triedy *LargeBlock*. Výpočet ostrosti všetkých ostatných riadkov vykonáva funkcia *CountRow* pomocou efektívnejšieho postupu(viď. 2.1.2). Napriek tomu, že trieda *S1* objekt *MagnitudeSpectrum* nijak nevyužíva, vytvára jej instanciu a predáva ju objektom *LargeBlock*, ktorý *MagnitudeSpectrum* využíva na počítanie ostrosti bloku. Tento postup je zvolený z dôvodu, že pri vytváraní(v konštruktore) *MagnitudeSpectrum* sa vykonávajú časovo náročné procesy ktoré inicializujú datové štruktúry, využívané pri počítaní ostrosti blokov.

LargeBlock

Objekt *LargeBlock* reprezentuje blok 32x32 pixelov a počíta jeho ostrosť. Pre počítanie ostrosti bloku slúžia viaceré metódy. Líšia sa v tom, aké optimalizácie na výpočet ostrosti využívajú. Metóda *CountFirst* nevyužíva žiadne optimalizácie a počíta ostrosť len prvého bloku, prvého riadka.*CountFirstInColumn* sa využíva na výpočet ostrosti blokov prvého riadka. Využíva informácie o ostrosti bloku bezprostredne vľavo. Na výpočet ostrosti blokov prvých v riadku slúži metóda *CountFirstInRow*. Optimalizácie tejto metódy sa odvíjajú od znalosti ostrosti bloku bezprostredne nad práve počítaným blokom. Ostrosti všetkých ostatných blokov sú počítané pomocou metódy *Count*. Táto metóda využíva najsilnejšie optimalizácie. Na reprezentáciu hodnôt max, min, mean pre pre všetky podbloky(viď. 2.1.2) slúži objekt *MMMData*.

MagnitudeSpectrum

Účelom triedy *MagnitudeSpectrum* je spočítať ostrosť bloku tak ako to popisuje článok [2]. Na výpočet ostrosti využíva objekty *ReadialFrequency*, *LinearRegression* a *FFT*, ktoré inicializuje v konštruktore. Výpočet ostrosti vykonáva metóda *GetSlope*, ktorej parametre určujú nad akými pixelmi má pracovať. V prvom kroku sa pomocou objektu *FFT* spočíta 2D fourierová transformácia daného bloku. V druhom kroku sa určí funkcia spádu intenzity fourierovej transformácie

od stredu ku krajom(viď 2.1.2). Nakoniec sa pomocou *LinearRegression* odhadne spád navzorkovanej funkcie.

FFF

Trieda *FFT* má jedinú verejnú metódu *Process*, ktorej sa argumentmi určí vstupné dvojrozmerné pole(pole pixelov, štartovný index). Výsledok transformácie sa uloží do verejného atribútu *m_ppData*, ktorý obsahuje dvojrozmerné pole komplexných čísel. Kvôli predchádzaniu zbytočnej alokácie a dealokácie je počas celého behu programu vytvorený len jeden objekt typu *FFT*, ktorý v konštruktore naalokuje *m_ppData*(v deštruktore dealokuje). V konštruktore sa teda musí dopredu pevne určit veľkosť vstupnej/výstupnej funkcie. To nám ale nevadí, kedže všetky bloky majú dopredu známy rozmer(32x32). Do naalokovaného priestoru pre výstupnú transformáciu sa opakovane zapisujú výsledky všetkých blokov. Pre výpočet fourierovej transformácie bol vybraný najefektívnejší známi algoritmus *fast fourie transform* [5], ktorý pracuje v čase $n \log(n)$.

LinearRegsession

LinearRegsession je jednoduchá trieda ktorá sa obmedzuje na počítanie lineárnej regresie v dvojrozmernom priestore. Navyše funkcii *GetAlfa* sa predávajú iba ypsilonové hodnoty bodov. Xové súradnice bodov sú implicitne odvodené z podaria(1 ... n). Tieto obmedzenia nám umožňujú spraviť veľmi jednoduché(implementačne aj výpočtovo) riešenie navrhnuté pomocou metódy najmenších štvorcov.

RadialFrequency

Úlohou triedy *LinearRegsession* je počítať vzdialenosť pixelu od stredu v bloku 32x32(LargeBlock). Tieto vzdialenosti sa využívajú pri určovaní poklesu intenzity *fft* od stredu ku krajom(obrázok 2.1.2). Kedže vzdialenosti majú rovnaké hodnoty pre každý blok, sú počítané len raz v konštruktore. Následne sú uložené vo verejnej položke *m_ppDistance*. Podobne ako objekt *FFT* aj *RadialFrequency* je vytvorený len raz počas celého behu programu.

4. Používateľská dokumentácia

Táto kapitola je venovaná používateľskej dokumentácii. Samotná aplikácia je veľmi jednoduchá a bola vytvorená za účelom preukázať funkčnosť segmentácie.

4.1 Spustenie

Pri spustení spustitelného súboru "SegmentationFrom.exe"je potrebné, aby sa v tom istom priečinku nachádzala knižnica "Geos.dll". Spustitelný súbor je spustitelný jedine na operačnom systéme Windows(xp a novší).

4.2 Vstupný formát

Po spustní aplikácie je potrebné ako prvé vybrať obrázok na segmentovanie zo súborového systému. Obrázok musí byť v formáte .JPG alebo .TIF (3 alebo 4 byt/pixel). Obrázok môže byť v ľubovolnom rozlíšení(odporúčané max 20 Mpix).

4.3 Segmentačný formulár

Obrázok určený k segmentácii vyplní formulár 1:1. V prípade vstupného obrázku s príliš vysokým rozlíšením sa formulár naškáluje na dopredu danú maximálnu šírku a výšku(1000, 700). Kvôli 100% vyplneniu formuláru obrázkom nie je možné výšku a šírku formulára škálovať. Formulár obsahuje v hornej časti ovládacie prvky, pomocou ktorých sa program ovláda.

File

V záložke *File* sa nachádzajú ovládacie prvky na načítanie a ukladanie obrázkov. Ukladať obrázky je možné v .JPG, .BMP alebo .PNG formáte. V prípade .PNG formátu bude oblasť pozadia plne priehľadná farba pomocou alfa kanálu. V prípade .JPG a .BMP bude farba pozadia špecifikovaná uživateľom. Ukladať je možné len v prípade, že proces segmentácie prebehol úspešne dokonca.

Parameters

Pomocou záložky *Parameters* nastavujeme parametre samotnej segmentácie. Obsahuje :

- typ segmentácie (Sharp[default], SharpAndColor, Color)
- farba pozadia (uplatnená jedine v .JPG a .BMP)

- privolá formulár pre editovanie parametrov (*EditForm*)

Select

Záložka *Select* slúži k interaktívnemu vyznačeniu popredia a pozadia. Po vybraní konkrétneho stavu, myšou vyznačíme popredie/pozadie na obrázku. Pre odstránenie vyznačených území je potrebné opätovné načítanie obrázku zo súborového systému.

Start/Stop

Tlačidlo *Start*/*Stop* slúži k naštartovaniu/ukončeniu segmentácie. Počas procesu segmentácie sú zablokované všetky ovládacie prvky(okrem *Stop* tlačidla). Po úspešnom skončení sa pôvodný obrázok nahradený vysegmentovaným. V prípade predčasného ukončenia sa celý proces zruší a aplikácia sa vráti do stavu pred začiatkom segmentácie. Pri opakovanom spustení začína proces odznova(nie je o nič efektívnejší). Po úspešnom skončení sa zobrazí nové tlačidlo(*Switch*) pomocou ktorého prepíname zobrazované obrázky.

Obr. 4.1: SegmentationForm

4.4 Editačný formulár

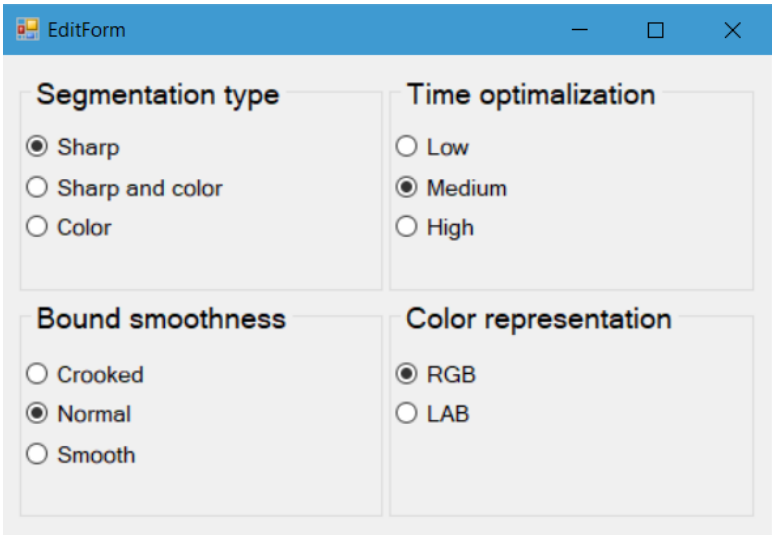
Editačný formulár slúži na editovanie štyroch segmentačných parametrov.

Segmentation type: určuje typ segmentácie(podľa akého predpokladu segmentujeme).

Time optimalization: nastavuje počet počítaných energií a masiek (2, 4, 6).

Bound smoothness: ovláda členitosť hranice medzi popredím a pozadím(pomocou parametrov θ).

Color representation: určuje farebnú reprezentáciu farieb pri modelovaní pomocou Gaussianovho zloženého modelu (uplatnené len pri segmentácii podľa farby a interaktívnej segmentácii).



Obr. 4.2: Editačný formulár

5. Experimenty

Táto kapitola sa venuje porovnaniu výsledkov našej implementácie s existujúcimi riešeniami. Testovať budeme len automatickú segmentáciu podľa ostrosti, kedže práve táto časť bola naším najhlavnejším cieľom. Ako vhodný program na porovnávanie bol zvolený Adobe Photoshop s funkciou Äuto focus", kedže práve táto funkcia rieši rovnaký problém(automatickú binárnu segmentáciu podľa ostrosti). Funkcia *Auto focus* bola zverejnená v roku 2014. V januári tohto roku(2018, v tomto čase bola dokončovaná naša implementácia) bola však zverejnená nová funkcia "Select Subject"ktorá rieši rovnaký problém avšak tentokrát pomocou AI(artificial inteligence)[12]. Porovnávať budeme všetky 3 implementácie.

5.1 Konfigurácia

Všetky testy boli vykonávané na rovnakej dopredu prednastavenej konfigurácii

Segmentation type: Sharp

Time optimalization: Medium

Bound smoothness: Normal

Color representation: Rgb (irelevantné)

Interaktívna pomoc: -

5.2 Dataset

Dataset obsahuje 31 testovacích obrázkov. Pre čo nejvernejšie testovanie boli vybrané fotografie zachytávajúce rôzne objekty(ľudia, zvieratá, autá ...), v rôznych podmienkach. Takisto boli použité fotografie s rôznymi rozlíšením naprieč používaného spektra(HD - 4k). Všetky testovacie obrázky spĺňajú invariant "zaostrené popredie, nezaostrené pozadie". Testovacie obrázky boli vybrané tak, aby tiež poukázali na nie vždy správnosť našej heuristiky(neostré pixely obklopené ostrými patria do objektu).

5.3 Výsledky

Výsledky boli porovnané s ručne vysegmentovanými vzormi(pixel po pixeli). V každom testovacom obrázku nás zaujímala percentuálna zhoda priradenia pixelu do popredia/pozadia s ručne vysegmentovaným vzorom. Celkový výsledok

nám určila priemerná percentuálna zhoda. Priemerná zhoda našej implementácie(*Geos*) **95.85** % je porovnatelná s priemernou zhodou funkcie *Auto focus* od Adobe **93.79** %. Čerstvá novinka *Select subject* avšak poráža obe implementácie s veľkým náskokom **99.17** % zhodou, ktorá sa blíži k úrovni chyby merania. Na testovaných obrázkoch sa ukázalo, že vlastné finálne úpravy pravdepodobnostnej mapy *S3* (2.1.4) nemusia vždy viesť k lepším výsledkom(test 8, 13, 28). Vo väčšine prípadov sa však ukázali byť užitočné.

5.4 Zlyhania

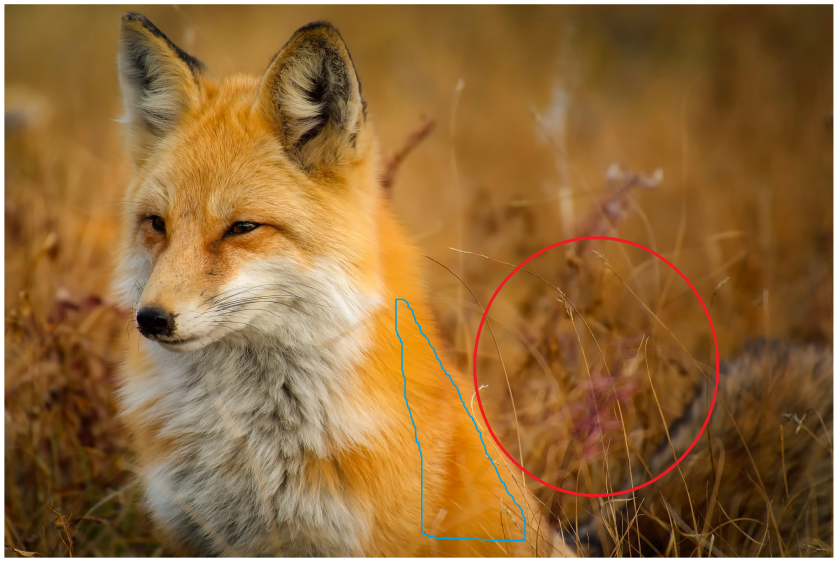
Aj keď priemerná úspešnosť našej implementácie na zvolenom datasete činila **95.85** %, pár testov sa od tejto úspešnosti výrazne vychyluje(4: 80%, 25: 88.94%, 29: 82.70%). Preto by sme chceli tieto zlyhania odôvodniť.

Test 4 mal zo všetkých testovaných obrázkov najnižšiu úpešnosť(80%). Dôvody tak nízkej úspešnosti sú dva. **1)** Časť objektu nie je v skutočnosti dobre zaostrená. **2)** Na obrázku sa vyskytujú ostré hrany(steblá), ktoré sú dobre zaostrené, avšak do objektu nepatria. Tieto steblá boli vyhodnotené ako ostré. Následne pre oblasti nachádzajúce sa medzi týmito steblami sa uplatnila naša heuristika 2.1.4(ak je objekt obklopený ostrými pixelmi, sám je ostrý).

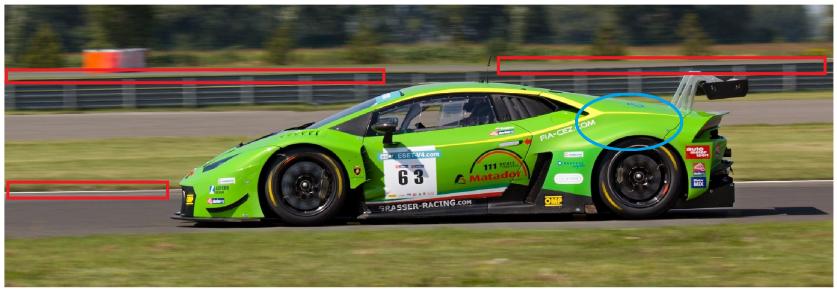
Test 25 s úspešnosťou 88.94 % opäť patril k jedným z najhorších. Tentokrát je celý objekt dobre zaostrený. Problém spočíva v type povrchu objektu. Jedná sa o veľmi hladký povrch(karosériu) bez akýchkoľvek hrán či šumu. Algoritmus *S3* vyhodnotí tieto oblasti ako neostré. V tomto prípade by mala zafungovať naša vlastná úprava(2.1.4). Avšak ostré hrany sa nachádzaju aj v nezaostrených oblastiach. Táto kombinácia vlastností segmentovaného obrazu vedie k nepresnému určeniu mapy ostrosti.

Test 29 mal druhú najnižšiu úspešnosť(82.70 %). V tomto prípade je spôsobená rozsiahlymi nezaostrenými oblasťami v objekte. Presne tento pomerne často sa vyskyjúci problém sa snaží riešiť *S3* & *Color*, ktorá na danom teste dosahuje úspešnosť 91.02 %.

A



B



C

Obr. 5.1: **A)** Test číslo 4. Červený kruh poukazuje na zaostrené steblá. Modrá zóna na nezaostrené časti objektu. **B)** Test číslo 25. Obsahuje hladký dobre zaostrený povrch, ostré hrany v pozadí. **C)** Test 29. Obsahuje rozsiahle nezaostrené oblasti v objekte.

	Geos	Auto focus	Select subject
1.	99.60	83.35	99.77
2.	96.08	87.55	99.51
3.	95.87	96.81	99.75
4.	80.00	90.83	98.86
5.	97.60	96.70	99.38
6.	99.12	99.31	99.10
7.	97.63	99.30	99.35
8.	95.70	99.28	99.28
9.	99.48	98.28	99.53
10.	91.79	90.96	98.03
11.	95.77	97.31	99.59
12.	97.64	98.10	99.09
13.	94.89	95.77	99.77
14.	99.71	99.62	99.90
15.	99.88	99.88	99.81
16.	99.54	94.84	99.68
17.	96.69	98.28	99.24
18.	97.65	96.73	98.41
19.	97.58	98.15	97.92
20.	98.52	59.54	99.56
21.	98.31	98.75	99.57
22.	99.40	99.51	99.67
23.	98.55	98.69	98.69
24.	95.60	94.53	98.12
25.	88.94	91.57	98.23
26.	94.78	91.99	99.57
27.	98.40	98.79	99.28
28.	97.79	98.62	99.43
29.	82.70	98.18	99.33
30.	89.82	57.78	98.34
31.	96.18	98.45	98.45
priemer	**95.85**	**93.79**	**99.17**

Záver

Cieľom práce bolo navrhnúť a implementovať riešenie automatickej binárnej segmentácie obrazu, na základe určitých predpokladov o obraze, doplenenú o interaktívnu segmentáciu. K dosiahnutiu tohto cieľa sme sa dopracovali skombinovaním postupov odvodených z odbornej literatúry. Prevzaté postupy sme ďalej doplnili o vlastné algoritmy, motivované vlatnými heuristikami a pozorovaniami. Počas vývoja projektu sa ukázalo, že najvhodnejšia cesta, dosahujúca najlepšie výsledky, je segmentácia podľa ostrosti(*S3*). Preto sa tomuto riešeniu venovala najväčšia pozornosť a dôraz.

Proces výpočtu sa nám rozložil na dve zložky.

1. detekcia ostrosti

2. segmentácia na základe pravdepodobnostnej mapy (mapy ostrosti)

Detekcia ostrosti je založená na článku [2]. Našou úlohou bolo si daný článok naštudovať, pochopiť použitým algoritmom a na záver ich efektívne implementovať. Samotná informácia o ostrosti oblastí v obraze môže byť takisto zaujímavá. Z dôvodu vyžívania detekcie ostrosti ako samostatného nástroja bol *S3* algoritmus vyvíjaný ako samostatný projekt, ktorý bol neskôr začlenený do projektu *Geos*.

Segmentácia na základe pravdepodobnosti je kvôli časovej efektivite odvodená z článku [4]. Účelom použitého riešenia(*Geos*) je dosiahnuť podobné výsledky ako známe riešenie pomocou minimálneho rezu, efektívnejším a jednoduchšie paralelizovatelným algoritmom. Použité riešenie nám umožňuje lepšie využiť výpočtový výkon moderných viac jadrových procesorov, s účelom segmentovať obrázky s vysokým rozlíšením.

Popísaná hlavná štruktúra projektu je rozšírená o alternatívne možnosti získavania pravdepodobnostnej mapy a o interaktívnu segmentáciu. Výsledky automatickej farebnej segmentácie sa avšak neukázali byť dostatočne uspokojujúce. Výsledky rozšírenia *S3* & *Color* sa ukázali byť rozumné, avšak požadované predpoklady na segmentovaný obráz sú príliš vysoké(ostrosť a farba) a tým toto riešenie nie je dostatočne všobecné. Samotná interaktívna segmentácia nebola od začiatku pre náš projekt zaujímavá, keďže dnes už existuje mnoho aplikácii založených na interaktívnej segmentácii, ktoré fungujú s vysokou presnosťou. Z týchto dôvodov bola automatická binárna segmentácia podľa ostrosti hlavným a aj dosiahnutým cieľom. Pre podloženie našej subjektívnej mienky, že táto segmentácia dosahuje rozumných výsledkov, boli vykonané experimenty. Tridsaťjeden ručne vysegmentovaných obrázkov bolo porovnaných s výsledkami našej automatickej segmentácie. Pre lepšie porovnanie bola do testu zahrnutá aj funkcia z programu Adobe Photoshop nazývaná Auto focus. Výsledky spočiatku naznačovali, že náš

program dokáže konkurovať vo funkcii automatickej binárnej segmentácie podľa ostrosti aj takým programom ako je Adobe Photoshop. V tomto januári(2018) Adobe vydal novú funkciu *Select subject* ktorá dokáže vykonávať automatickú binárnu segmentáciu bez akýchkoľvek predpokladov a napriek tomu dosahuje omnoho vyššiu úroveň správnosti.

Na záver pre vhodné a príjemné testovanie funkčnosti knižnice bola vytvorená desktopová aplikácia s grafickým rozhraním. Pôvodné rozhranie knižnice ktoré obsahovalo absolútne cesty k vstupnému/výstupnému obrázku bolo pozmenené pre potreby aplikácie.

Zoznam použitej literatúry

[1] Lina J. Karam S. Alireza Golestaneh. Spatially-varying blur detection based on multiscale fused and sorted transform coefficients of gradient magnitudes. *Journal of the American Statistical Association*, 53(282):457–481, 2008.

[2] Damon M. Chandler. Cuong T. Vu, Thien D. Phan. A spectral and spatial measure of local perceived sharpness in natural images. *IEEE*, 34(2):187–220, 2013.

[3] Image segmentation using minimum st cut. http://cmp.felk.cvut.cz/cmp/courses/33DZOzima2007/slidy/mincut+maxsum-Werner.pdf. Accessed: 2017-10-30.

[4] Toby Sharp Antonio Criminisi. Geodesic image segmentation. *Microsoft Research, Cambridge, UK*, 49:309–319, 2006.

[5] Discrete fourier transform. fast fourier transform. http://paulbourke.net/miscellaneous/dft/. Accessed: 2017-10-30.

[6] Gaussian mixture model. https://brilliant.org/wiki/gaussian-mixture-model/. Accessed: 2017-10-30.

[7] Sigmoid function. https://en.wikipedia.org/wiki/Sigmoidfunction. Accessed: 2017-10-30.

[8] Using faster exponential approximation. https://codingforspeed.com/using-faster-exponential-approximation/. Accessed: 2017-10-30.

[9] Pekka J. Toivanen. New geodesic distance transforms for gray-scale images. *Lappeenranta University of Technolog*, 14(2):14, November 1994.

[10] How-to: Re-encode a jpeg image with metadata. https://msdn.microsoft.com/en-us/library/windows/desktop/ee719794(v=vs.85).aspx. Accessed: 2017-10-30.

[11] Default marshaling behaviors. https://docs.microsoft.com/en-us/dotnet/framework/interop/default-marshaling-behavior. Accessed: 2017-10-30.

[12] Select subject is here: Photoshop now has ai-powered one-click selections. https://petapixel.com/2018/01/23/select-subject-photoshop-now-ai-powered-one-click-selections/. Accessed: 2017-10-30.