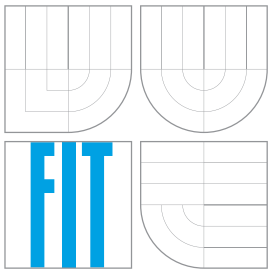
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

PROTOTYPOVÁNÍ FOTOGRAFICKÉ KOMPOZICE POMOCÍ ROZŠÍŘENÉ REALITY

PROTOTYPING OF PHOTOGRAPHIC COMPOSITION USING AUGMENTED REALITY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR PALATA

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. ADAM HEROUT, Ph.D.

BRNO 2015

Abstrakt

Cílem této práce bylo vytvoření aplikace, díky které by uživatel mohl vytvářet v několika jednoduchých krocích zajímavé a realisticky vypadající fotomontáže za pomoci mobilního telefonu. Důraz je kladen na jednoduchost aplikace, uživatelskou přívětivost a rychlost pořízení výsledné montáže. Práce diskutuje možné přístupy k vytvoření takové aplikace, algoritmy k tomu potřebné a problematiku implementace algoritmů počítačového vidění na mobilních zařízeních. Dále diskutuje možnosti urychlení algoritmů pomocí paralelizace na novějších zařízeních s operačním systémem iOS.

Abstract

The goal of this work was to create an application providing the user with an option of creating interesting and realistic looking montages in few easy steps using a mobile phone. The focus is on simplicity, user friendliness and the speed with which the final montage can be obtained. The work discusses available approaches to building such application, the algorithms needed and also the problems with the implementation of computer vision algorithms on a mobile device. It also discusses possible options for speeding up the algorithms through parallelization on newer devices running the iOS operating system.

Klíčová slova

Segmentace, matting, fotografická kompozice, mobilní aplikace, iOS, rozšířená realita

Keywords

Segmentation, Matting, Photographic Composition, Mobile Application, iOS, Augmented Reality

Citace

Petr Palata: Prototypování fotografické kompozice
pomocí rozšířené reality, diplomová práce, Brno, FIT VUT v Brně, 2015

Prototypování fotografické kompozice pomocí rozšířené reality

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana docenta Adama Herouta.

.....
Petr Palata
27. května 2015

Poděkování

Rád bych poděkoval panu docentu Adamu Heroutovi za jeho cenné připomínky a odborné rady při vypracování této diplomové práce.

Obsah

Kapitola 1

Úvod

Mobilní telefony jsou nedílnou součástí každodenního života většiny obyvatel civilizovaných zemí. V období uplynulých let tato kapesní zařízení prošla bleskovým vývojem, nezvykle rychlým i s přihlédnutím k běžné strmosti technologických křivek. Z neforemné pomůcky na vyřizování hovorů na cestách se mobilní telefon změnil na plnohodnotné výpočetní zařízení s výkonem srovnatelným s počítači před několika málo lety. Mobilní telefony už navíc nejsou jenom výpočetním, ale také multimediálním zařízením obsahující kameru, fotoaparát nebo výjimečně stereokameru. Díky jejich výkonu i zmíněnému vybavení existuje již nespočet aplikací umožňujících jak pořizování, tak editaci multimediálního obsahu.

Tyto aplikace slouží z velké části pro zábavu uživatelů a umožňují sdílení upravených fotografií nebo videí na sociálních sítích. Zajímavým doplňkem k těmto aplikacím a zároveň elegantním použitím poznatků počítačového vidění a rozšířené reality by mohla být aplikace umožňující uživateli vytvářet interaktivní montáž do jiné fotografie.

Taková aplikace by se dala využít nejen pro zábavu, ale také jako profesionální nástroj fotografů pro vytváření předběžné představy o fotografické kompozici ve scéně. Postupem vytvoření takovéto aplikace, použitými algoritmy počítačového vidění a problémy s implementací těchto algoritmů na mobilních zařízeních se zabývá tato práce.

Kapitola 2

Rozšířená realita a virtualita

Zkráceně lze říci, že rozšířená realita je implementace algoritmů počítačového vidění, které pomocí kamery a zobrazovacího zařízení (displeje) dokáží vytvořit věrohodný obraz snímané scény obohacený (rozšířený) o další objekty. Děje se tak za pomocí míchání obrazu reálného světa obdrženého nahrávacím zařízením (v tomto konkrétním případě mobilní kamerou) s počítačově vytvořenou scénou. Existuje rozdělení typů rozšířené reality podle snímací a zobrazovací technologie. Mobilní telefony patří do kategorie video see-through, při kterém je výsledná scéna uživateli prezentována pomocí displeje a navíc se pro pořízení obrazu reálné scény používá kamera.

2.1 Aplikace v pojetí Milgramovy definice rozšířené reality

Milgramova [9] definice rozšířené reality spočívá v představení reality-virtuality continuum, které je rozděleno na čtyři části a to podle množství uměle vytvořené informace. Pojem realita není potřeba vysvětlovat, jedná se pouze o reálnou scénu, která například v případě brýlí ani nemusí být zobrazená přes prezentační zařízení. Na opačném straně kontinua je virtuální realita, ve které je uživateli prezentována kompletně uměle vytvořená scéna. Uprostřed kontinua je pak část, která je souhrnně nazvána jako kombinovaná realita a zapouzdřuje v sobě dva pojmy. Prvním pojmem je rozšířená realita, která říká, že ve výsledné scéně má ještě pořád většinový podíl reálná informace. U rozšířené virtuality je tomu přesně naopak - výsledná scéna je z větší části virtuální. Aplikace, jejímž vytvářením se tato práce zabývá proto odpovídá spíše definici rozšířené virtuality, protože přídáváme pouze kousek reality (člověka) do již vytvořené scény (jiné fotografie). Pojem rozšířená virtualita je však používaný zřídka, proto se mluví o této aplikaci jako o rozšířené realitě.

Kapitola 3

Vývoj aplikací na platformě iOS

Jelikož má být cílová aplikace implementována na mobilní platformě, je vhodné zmínit se o možnostech a úskalích zpracování obrazových dat na platformě iOS. Tato kapitola popisuje hlavní stavební kameny práce s obrázky, vestavěné knihovny a poskytované programátorské rozhraní. V poslední řadě je také diskutován samotný proces publikování hotové aplikace do AppStore.

3.1 Core Graphics (Quartz) a Core Image

V iOS jsou dostupné dvě knihovny sloužící pro práci s obrazovými daty. Prvním je Core Graphics [18], který obaluje správu barev a umožňuje vykreslování grafických primitiv a cest (angl. paths) do grafických kontextů. Tento framework by měl být použit pokaždé, když se mění tovární kód pro vykreslování uživatelského rozhraní. Kontext však nemusí být pouze obrazovka telefonu (v iOS nazývána jako View), ale může to být také obrázek. Většina operací tohoto frameworku je pak implementována přímo nad kontexty, takže se pracuje v obou případech úplně stejně.

Druhým pro tuto práci důležitým frameworkem je Core Image [19], který v sobě zapouzdřuje širokou škálu operací a filtrů pro zpracování a analýzu obrazových dat. Core Image navíc nabízí možnost implementovat uživatelské filtry, které poté mohou běžet na grafickém procesoru.

3.2 Publikování aplikací na AppStore

Aby mohla být aplikace publikována na AppStore (internetový obchod s aplikacemi pro zařízení firmy Apple), musí projít schvalovacím procesem. Při tomto procesu se kontroluje, jestli aplikace splňuje požadavky společné pro všechny publikované aplikace. Proces kontroly odebírá vývojářum určitou svobodu v implementaci, ale na druhou stranu zaručuje konzistenci grafické úpravy a dobrých zvyků. Jmenovitě se musí dodržovat [20] a také [17].

Při nedodržení pravidel z těchto dokumentů je ve většině případů aplikace odmítnuta, ale na druhou stranu vývojář také obdrží důvody odmítnutí a může je díky tomu jednoduše opravit. Kontrola je do určité míry subjektivní a proto se může stát, že aplikace je z nějakého důvodu odmítnuta i když vývojář dodrží všechna pravidla. Aplikace v této práci, tak jak je navržena, by neměla přímo porušovat žádná pravidla pro odeslání na AppStore, takže pravděpodobnost objektivního odmítnutí by měla být minimalizována.

Kapitola 4

Segmentace a matting

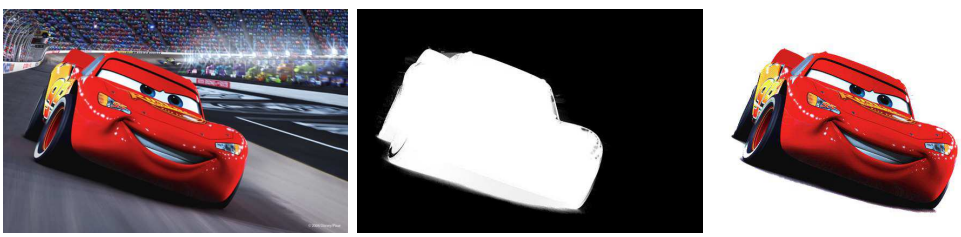
Hlavní část vytvářené aplikace je výřez člověka ze scény. Obecně však vždy nemusí jít o člověka, ale může to být libovolný jiný objekt. Takové obecné operaci vyřezání objektu z pozadí se říká cutting nebo segmentace. Vyřezávání popředí je známo hlavně z průmyslu filmových efektů, kde se běžně používá při klíčování na zelenou nebo modrou barvu pro následné doplnění upraveného pozadí. Segmentace se však nemusí provádět jen za pomoci klíčování na určitou barvu, ale lze použít pokročilejších modelů pro popředí a pozadí pro odhadnutí požadovaného výřezu. K této operaci pak slouží celá řada algoritmů, z nichž některé budou popsány v této kapitole.

Operace výřezu popsaná dříve se zabývá pouze přiřazením jednotlivých pixelů do pozadí nebo popředí. Výsledkem takového přiřazení je výřez popředí, který má však ostré hrany. Vložení takového výřezu do scény by nevedlo k uvěřitelnému zobrazení. Z tohoto důvodu následuje po cuttingu operace matting. Matting se snaží nalézt ideální hodnoty průhlednosti (hodnoty α) na okraji výřezu s ohledem na co nejrealističtější montáž do scény. Většina moderních algoritmů na cutting se zabývá i mattingem, takže není třeba tyto dva pojmy dále rozlišovat a můžou být souhrnně nazývány jako vyřezávání obrazu. Na obrázku 4.1 je zobrazen celý proces od vyříznutí po vložení obrázku do výsledné scény [6].

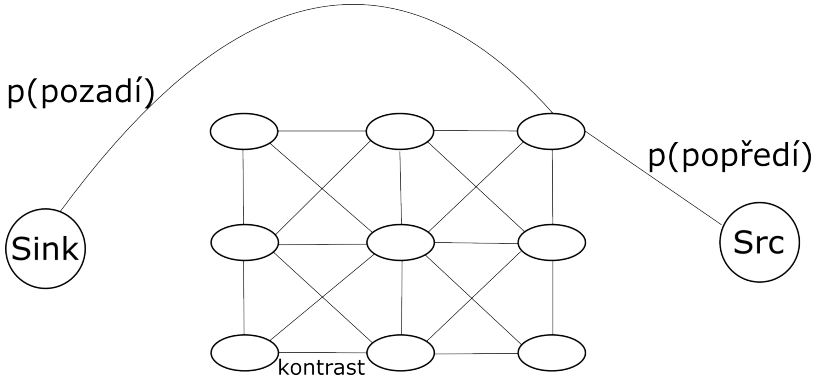
4.1 GraphCut

GraphCut [3] je algoritmus na segmentaci, který byl vytvořen kvůli potřebě pro algoritmus na jednoduché vyřezání popředí s důrazem na co nejmenší uživatelskou náročnost a co nejmenší počet uživatelských vstupů.

Inicializace algoritmu probíhá označením částí obrázků pomocí „štětce“ jako popředí a jako pozadí. Z této inicializace je pak možno natrénovat 2 modely - jeden pro popředí a jeden pro pozadí. Modely jsou kombinací gaussovských rozložení (angl. Gaussian Mix-



Obrázek 4.1: Proces získání alfa masky



Obrázek 4.2: Sestrojený graf pro vstup algoritmu min-cut max-flow

ture Models — GMM). Po natrénování modelů se vypočítá pravděpodobnost náležitosti jednotlivých pixelů do popředí a do pozadí. Tyto pravděpodobnosti jsou následně použity k minimalizaci energie zapsané následujícím vzorcem:

$$E = \sum p(x)_P + \sum_S kontrast(p1, p2)$$

kde P je množina všech pixelů a S je množina všech dvojic sousedních pixelů.

Minimalizace se provádí tak, že se vytvoří graf, kde každý pixel odpovídá jednomu uzlu. Vytvořený graf má ale navíc dva speciální uzly, v odborné literatuře nazývané jako source (zdroj) a sink (stok). Tyto dva uzly představují náležitost jednotlivých pixelů k popředí a pozadí. Náležitost je označena hranami vedoucími do těchto uzlů a hrany jsou ohodnoceny předešle vypočtenými pravděpodobnostmi. Poté se do grafu přidají hrany mezi pixely (čtyř nebo osmikolí každého pixelu) a těmto hranám se přiřadí hodnota nazvaná jako kontrast. Kontrast mezi dvěma pixely se vypočítá jako:

$$kontrast(p1, p2) = \|p1 - p2\|^2$$

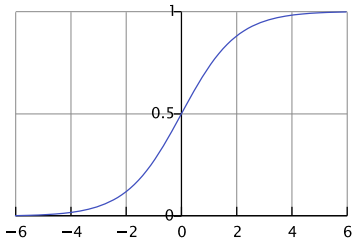
Ilustrace výsledného grafu je na obrázku 4.2.

4.2 Grabcut

Myšlenku co nejmenšího uživatelského vstupu posouvá ještě o něco dále algoritmus Grabcut [12]. Narozdíl od předchozího algoritmu se inicializuje pouze obdélníkem a není třeba přesně vyznačovat pixely popředí a pozadí v trimapě. Prvotní postup získání výřezu je stejný jako u algoritmu Graphcut. Hlavním rozdílem je, že postup získání popředí a trénování modelů se aplikuje iteračně, takže uživatel obdrží stejně přesný výsledek bez nutnosti pracného označení. Algoritmus navíc po dokončení a prezentaci výřezu uživateli umožňuje explicitně označit pixely popředí nebo pozadí, v případě že segmentace nedopadla podle očekávání. Uživatel tak může provést menší množství úprav, které povedou k mnohem přesnějšímu výsledku.

4.2.1 Naivní matting

GrabCut kromě iterativní aplikace algoritmu GraphCut představuje i naivní postup, jak provést matting výsledné segmentace pro zvýšení realističnosti. Princip tohoto přístupu



Obrázek 4.3: Sigmoidální funkce

je zvětšení okraje výsledné segmentace o určitý počet pixelů rovnoměrně po celé délce okraje. Na obrazové body, které vznikly z tohoto rozšíření, je pak směrem zevnitř objektu aplikována sigmoidální funkce (ilustrace na obrázku 4.3, která patřičně vyhladí α hodnoty a dodá tak na realističnosti vzhledu výsledného objektu.

4.3 Background cut

Oba předešlé algoritmy se zabývaly pouze jednotlivými obrázky a ne jakoukoliv jejich sekvencí. Tímto problémem se zabývá algoritmus Background cut [14]. Základ algoritmu tvoří předpoklad, že před začátkem vyřezávání z videosekvence máme obrázek pozadí, ze kterého můžeme natrénovat úvodní model pozadí. Stejně jako v případě Graphcutu se provádí v každém snímku minimalizace pomocí min-cut max-flow algoritmu. První rozdíl oproti Grabcutu je způsob trénování modelu pozadí. Pro pozadí se modely trénují hned dva. Jeden je pouze globální gaussovské rozložení v RGB prostoru a v témže prostoru je natrénován i druhý model — GMM. Výsledná pravděpodobnost náležitosti do pozadí je potom součet pravděpodobností obdržených z těchto dvou modelů. Váhy jednotlivých modelů se upravují koeficienty, jejichž součet musí být jedna.

Druhým rozdílem je obstarání pixelů pro trénování GMM modelu popředí. Jelikož máme dostupné již od prvního snímku sekvence barevné modely pozadí, můžeme pro každý pixel vypočítat pravděpodobnost náležitosti do pozadí. Z vypočtených pravděpodobností se sestrojí trimapa následujícím způsobem. Zvolí se dvě hodnoty prahu — nižší a vyšší hodnota. Poté se pravděpodobnosti porovnají s jednotlivými prahy a přiřadí se jim náležitost podle následujícího klíče

$$trimap(x) = \begin{cases} B & \text{pokud je x menší než nižší práh} \\ F & \text{pokud je x větší než vyšší práh} \\ U & \text{jinak} \end{cases}$$

Vytvořením takové trimapy vznikne i množina pixelů popředí. Z těchto pixelů se následně natrénuje GMM model popředí. Jelikož je tímto krokem získán model popředí, může být sestrojena část grafu s pravděpodobnostmi ke zdroji a stoku. Výrazná změna je i ve výpočtu kontrastu, který je nyní adaptivní podle velikosti změny prvotního obrázku pozadí a současného pozadí. Algoritmus se totiž snaží eliminovat veškerý kontrast u pixelů pozadí z důvodu snadnějšího vyříznutí pixelů popředí.

4.4 Interactive Cutout

Dalším algoritmem využívajícím informace dostupné ve videu je Interactive Cutout [15]. Jako předchozí zmíněné algoritmy je i tento založen na minimálních řezech v grafu. Výhodou

tohoto algoritmu je, že samotná segmentační část dokáže běžet na videu v reálném čase pouze s malým uživatelským vstupem (několik tahů štětcem pro pozadí a popředí).

Tento algoritmus má z hlediska zpracování v reálném čase jedno úskalí. Video, které je na vstupu aplikace musí projít časově náročným předzpracováním. Toto zpracování je plně automatické, takže z hlediska zátěže uživatele nepřidává nic navíc. Problémem tohoto předzpracování je tak pouze jeho časová náročnost. Předzpracování je tvořeno konvertováním videa do hierarchické grafové reprezetance, kterou pak efektivně dokáže zpracovat přizpůsobený algoritmus GraphCut. Konverze probíhá pomocí algoritmu mean-shift, který kóduje odlišnosti mezi snímky do speciální datové struktury.

Po úvodní segmentaci je navíc pro pořízení kvalitního výřezu nutno vystavit výsledné video dodatečnému automatickému zpracování, které je na procesorový čas stejně náročně jako úvodní předzpracování.

Z hlediska použití ve vytvářené aplikaci je tento algoritmus dobrou inspirací, protože však nahrávání videa, na kterém je v naší aplikaci prováděna segmentace, probíhá v reálném čase, nelze ho díky nutnosti předzpracování použít v celém jeho rozsahu.

4.5 GridCut

Možnostmi paralelizace často používaného algoritmu pro nalezení minimální toku v grafu se zabývá článek [5], který diskutuje možnosti efektivního použití paměťové cache pro snížení časové náročnosti výpočtu.

Pro paralelizaci využívá již existujícího přístupu [7, 2] a diskutuje možná vylepšení z hlediska efektivnější optimalizace paměťové lokality. Podle výsledků ale stále nedosahuje rychlosti potřebné pro zpracování v reálném čase.

Největším problém je však dostupnost tohoto algoritmu. Ačkoliv dosahuje dobrých výsledků, jedná se o produkt, který lze používat zdarma pouze pro nekomerční účely [24]. Jelikož se v budoucnu uvažuje o komercializaci aplikace, kterou se zabývá tato práce, nebyl zatím tento algoritmus použit. Zmíněné koncepty však mohou být hodnotnými při případné optimalizaci výkonu na mobilním zařízení.

4.6 Barevné histogramy

Všechny předchozí algoritmy přistupovaly k obrázku jako ke grafu, nad kterým provádíme nějakou optimalizační techniku (hledání minimálního toku). K segmentaci však lze přistupovat i jiným způsobem a to například jako k individuálním pixelům, které zvlášť klasifikujeme do tříd popředí nebo pozadí.

Můžeme tak například vytvořit 3D RGB histogram z uživatelem vyznačených pixelů a následně ve snímku pro každý RGB pixel vypočítat vzdálenost od tohoto histogramu a porovnat tento výsledek s prahem. Tímto způsobem lze zpracovat celý snímek pixel po pixelu a dostat tak výslednou segmentaci. Výhodou tohoto přístupu je i to, že klasifikační část je triviálně paralelizovatelná, protože mezi klasifikacemi jednotlivých pixelů neexistují žádné datové vazby.

Obyčejný 3D histogram má však pro malé množství vstupních pixelů problém s řídkostí a navíc tento naivní přístup není dostatečně robustní. Řešením těchto problémů, navíc při nižší složitosti výpočtu (vyšší rychlosti), je projekce jednotlivých pixelů na množinu přímek v RGB prostoru [16].

Mějme RGB prostor hodnot 0–255, ve kterém je rovnoměrně rozmístěno 13 přímek, kde každá z nich prochází bodem (128, 128, 128). Každá z těchto přímek reprezentuje 1D histogram. Místo 3D histogramu se tedy prostor redukuje na $13 \cdot poetbin$ 1D histogramu. Při klasifikaci pak můžeme vypočítat pravděpodobnost pixelu jako

$$p(x) = \sum_{i=1}^{13 \cdot b} x_i \cdot \frac{h_i}{N}$$

kde x_i je hodnota itého binu histogramu vzniklého konkatenací všech histogramových projekcí daného pixelu, N je počet trénovacích pixelů, b je počet binů 1D histogramu a h_i je hodnota itého binu histogramu natrénovaného z uživatelské anotace. Vypočtenou pravděpodobnost lze následně porovnat se zvoleným prahem a rozhodnout o přiřazení pixelu do popředí nebo do pozadí.

4.7 Přehled algoritmů pro matting

Všechny předešlé algoritmy se zabývaly pouze nalezením ostré segmentace popředí. Jak už bylo zmíněno, pro realističnost výsledné montáže musíme ještě provést matting. Matematicky řečeno matting řeší problém nalezení koeficientů α pro každý pixel obrázku v rovnici

$$I = \alpha F + (1 - \alpha)\, B$$

kde F je barevná složka popředí a B je barevná složka pozadí. Aby vůbec byla možnost nalézt hodnoty α, potřebujeme nejprve znát trimapu. Hodnoty pak počítáme v okolí přechodu mezi popředím a pozadím, kde je segmentace neznámá. Následuje přehled některých algoritmů na matting s jejich krátkým popisem.

- **Knockout** - výsledná barva je určena ze sousedních pixelů pomocí váhovaného průměru [1].
- **Bayesian matting** - seskupí barvy popředí a pozadí k několik středům, na ty natrénuje gaussovská rozložení a poté hledá takové hodnoty alpha, které mají maximální likelihood [4].
- **Poisson matting** - pro odhadnutí hodnot alpha nepoužívá statistických modelů, ale místo toho vypočítá gradient ze segmentace a poté vypočítá hodnoty matte pomocí řešení Poissonových rovnicí [13].

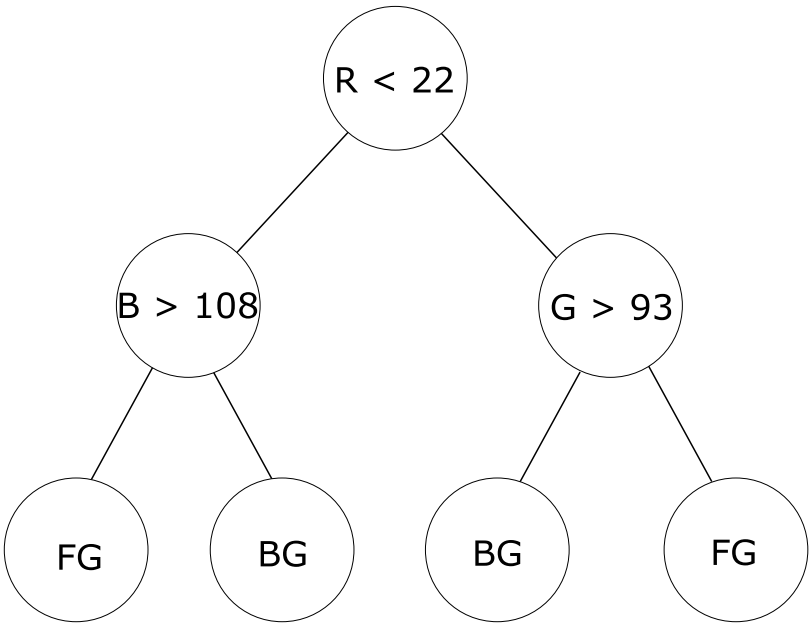
Kapitola 5

Rozhodovací stromy

Existuje mnoho technik počítačového učení, kterými se dá naučit binární klasifikátor. Jednou z těchto metod jsou rozhodovací stromy, které se používají v širokém spektru úloh počítačového vidění. Rozhodovací strom je klasický (často binární) strom, jehož specifičnost spočívá v tom, že se v každém uzlu nachází obecně různé kritérium, podle kterého se volí další průchod stromem.

Každý listový uzel poté náleží jednomu konkrétnímu koncovému rozhodnutí. Takové rozhodnutí může být teoreticky v jakémkoliv formátu, ale pro jednoduchost budeme v tomto případě uvažovat rozhodnutí mezi dvěma klasifikačními třídami. Rozhodnutí v koncových uzlech nemusí být unikátní, tedy uzel označující určité rozhodnutí se může na pozici listu ve stromě vyskytovat i vícekrát.

Ilustraci rozhodovacího stromu výšky tři, který slouží k rozhodnutí, jestli pixel náleží do popředí nebo do pozadí, lze vidět na obrázku 5.1.



Obrázek 5.1: Ukázka rozhodovacího stromu

5.1 Trénování rozhodovacích stromů

Aby bylo možno rozhodovací strom použít jako klasifikátor, musí se nejprve natrénovat některým z mnoha dostupných algoritmů. Jedním ze základních přístupů pro trénování rozhodovacích stromů je algoritmus ID3. V pseudokódu ho lze zapsat následovně

Algorithm 1 Trénování stromu

```
function TrainTree(Root U, Samples S)
    if All samples in S have the same label then
        U.isLeaf ← True
        U.label ← label
        return U
    end if
    for all split parameters P do
        g ← Gain(S, P)
        if bestGain < g then
            bestGain ← g
        end if
    end for
    U.left ← TrainTree(U.left, Sleft)
    U.right ← TrainTree(U.right, Sright)
end function
```

Dále existují rychlejší a efektivnější rozšíření tohoto algoritmu jako například C4.5 nebo C5.0, ale jedná se pouze o komerčně poskytované produkty.

5.2 Informační entropie

Informační entropie je nad konečnou množinou vzorků definována jako

$$H(S) = -\sum_{y} p_y log_2(p_y) \tag{5.1}$$

kde pro p_y je počet výskytu štítku i v množině vzorků S. Aby tato definice fungoval i nad prázdnou množinou vzorků S, pro potřeby této práce budeme definovat $0 \cdot log2(0) \equiv 0$.

5.3 Zisk (gain)

Kromě definice informační entropie je nutno definovat ještě další pojem, který je použit jako metrika při trénování rozhodovacích stromů. Jedná se o informační zisk (gain) při použití daného uzlu jako potomka. V základní formě je definován jako

$$Zisk(i) = H(S(i)) - \sum_{k \in \{P,L\}} \frac{\|S_k(i)\|}{\|S(i)\|} H(S_k(i))$$

kde H(S(x)) je výše zmíněná informační entropie a S(x) je množina vzorků uzlu.

5.4 Příklad natrénovaní stromu

Mějme množinu P vzorků, kde každý vzorek nese označení l z množiny L. Množina Y bude obsahovat označení fg označující, že vzorek náleží do popředí a označení bg říkající, že vzorek x náleží do pozadí. Nyní uvažujme, že množina X bude obsahovat 6 vzorků $\{12, -21, 18, 14, -13, 9\}$ s jejich korenspondujícími štítky $\{fg, bg, fg, fg, bg, fg\}$. Pokud budeme uvažovat, že rozhodování na uzlu bude probíhat jednoduchým porovnáním vzorku oproti zvolenému prahu a práh může nabývat hodnot z množiny $\{8, 16, -15\}$, potom trénování stromu nad množinou vzorků X podle algoritmu ID3 bude probíhat v následujících krocích.

- Vezme se vstupní množina X a spočítá se zisk pro všechny možné hodnoty prahu.
- Zjistí se, že největší informační zisk má uzel při prahu 8.
- Algoritmus se rekurzivně spustí nad levým a pravým potomkem.
- Levý potomek zjistí, že obsahuje pouze štítky fg a tedy označí svůj uzel za fg a ukončí.
- Pravý potomek naopak zjistí, že obsahuje pouze štítky bg a tedy označí svůj uzel za bg a ukončí.
- Rekurze se tím ukončí a algoritmus vrátí natrénovaný strom.

5.5 Náhodné rozhodovací stromy

Ačkoliv jsou rozhodovací stromy velmi silným nástrojem při klasifikaci, přináší i určité nevýhody. Problémem používání jednoho natrénovaného stromu je jeho tendence k přetrénování [11]. Pokud budeme navíc zkoušet pro trénování každého uzlu všechny kombinace, můžeme narazit na problémy spojené se složitostí při větším množství parametrů. Protože je složitost úzce spojena s rychlostí trénování, potřebujeme tomuto problému předejít.

Řešením je zavést do trénování stromů náhodnost dvěma kroky. Prvním krokem je vytvoření většího počtu natrénovaných stromů (takzvaného lesa), které budou při klasifikaci hlasovat. Tento krok sám o sobě však náhodnost do modelu nepřináší a pokud by nebyla do modelu jinak přidána náhodnost, vedl by pouze ke zvýšení složitosti výpočtu natrénováním několika identických stromů.

Do trénování se tedy zavádí náhodný výběr podmnožiny parametrů z množiny všech dostupných parametrů. Každý strom tím získá prvek náhody podle toho, které parametry jsou pro trénování jednotlivých uzlů zvoleny. Jednotlivé stromy jsou pak v rámci lesa unikátní a dodávají tak na robustnosti výslednému klasifikátoru.

Kapitola 6

Implementace rychlé segmentace

Součástí navrhované aplikace musí být algoritmus, který bude schopen na mobilním zařízení v reálném čase segmentovat objekt popředí, pouze na základě minimální anotace poskytnuté tahem prstu přes požadovaný objekt. Přístupy popsané v předchozí kapitole se soustředily spíše na dosažení co největší přesnosti než na dostupnost výpočtu v reálném čase. Proto bylo nutné navrhnout algoritmus, který by splňoval zmíněné požadavky a zároveň by dosahoval rozumné přesnosti.

Požadavek na přesnost nemusí tak striktní jako u algoritmů používaných v profesionálních grafických editorech, protože pro výslednou montáž lze provádět ještě další zpracování, které už nemusí probíhat v reálném čase. Příkladem takového zpracování může být napojení na některý z velmi přesných, ale pomalých algoritmů (GrabCut).

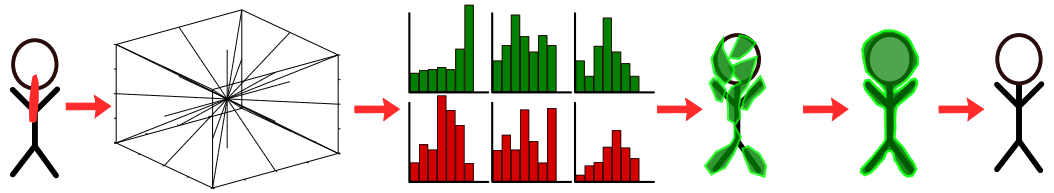
Tato kapitola obsahuje popis návrhu takového algoritmu založeného na souboru barevných histogramů získaných projekcí na přímku z RGB prostoru a jeho následné vylepšení pomocí metody náhodných rozhodovacích stromů. Je zde popsána implementace obou těchto přístupů v desktopovém prostředí.

6.1 Navržený systém

Základní podmínkou návrhu algoritmu a celého systému byl co nejmenší uživatelský vstup. Proto bylo rozhoduto, že celý systém bude závislý pouze na anotaci ekvivalentní k jednomu souvislému tahu prstem na obrazovce mobilního telefonu. Ostatní algoritmy pracují většinou na stejném principu, ale poskytují možnost dělat anotaci více tahy a navíc rozlišovat mezi anotací popředí a pozadí. Navržený algoritmus tedy omezuje uživatelský vstup na absolutní minimum.

Přehled systému je na obrázku 6.1. Jak už bylo řečeno, jako vstup algoritmu je rychlá uživatelská anotace objektu popředí pomocí tahu prstu. Následně je na tomto vstupu natrénován klasifikátor (obě verze klasifikátoru budou popsány v následujících kapitolách) a poté již může systém zahájit klasifikaci nad příchozími snímky. Samotná klasifikace by měla probíhat v reálném čase na videu přijímaném z kamery mobilního telefonu.

Tento návrh byl implementován ve dvou verzích, přičemž se liší pouze v použitém klasifikátoru. Oba vytvořené klasifikátory mají však stejný výstup (černobílou masku, kde černé pixely značí pozadí a bíle popředí) a v tomto návrhu tak mohou být zaměnitelné.



Obrázek 6.1: (1) Vstup algoritmu je uživatelská anotace. (2) Druhým krokem je vždy projekce pixelů na přímky (3) Natrénování klasifikátoru (histogramový model nebo rozhodovací stromy) (4) Výsledek segmentace

6.2 Naivní per—pixel přístup

V kapitole 4.6 byla popsána metoda redukce dimenzionality projekcemi na 13 přímek v RGB prostoru. Součástí této práce byla implementace tohoto přístupu a to nejprve jako prototyp ve skriptovacím jazyce Octave a jeho následná konverze do jazyka C pro snadnější integraci do mobilního zařízení.

Existuje více postupů, kterými se dá dosáhnout vytvoření projekce na přímku ve třídimenzionálním prostoru jako je prostor RGB. Prvním postupem je násobení vektoru reprezentujícího bod projekční maticí. Tu lze sestavit podle vzorce

$$P = e_1 \cdot e_1^T + e_2 \cdot e_2^T + e_3 \cdot e_3^T \tag{6.1}$$

kde e_1 je jednotkový vektor reprezentující přímku, na kterou děláme projekci a e_2 a e_3 jsou libovolné jednotkové normálové vektory této přímky, avšak navíc navzájem kolmé. Tyto tři přímky tak tvoří vektorovou bázi, pomocí níž lze maticovým násobením promítnout bod na zvolenou přímku.

Tento přístup je jednoduše implementovatelný, ale maticové násobení se stává přítěží, pokud se snažíme dosáhnout reálného času. Projekce lze však vypočítat i jinak a to pouze za pomoci několika sčítání a jednoho dělení. Navíc lze narozdíl od maticového násobení, které díky požadavku na jednotkové vektory obsahuje jiná než celá čísla, implementovat tento přístup bez použití čísel v plovoucí řadové čárce. Je totiž obecně známo, že celočíselné výpočty jsou několikrát rychlejší než výpočty v plovoucí řadové čárce.

Maticové násobení jde tedy nahradit rovnicí reprezentující projekci bodu x na přímku P

$$P(x) = p_1 \cdot x_1 + p_2 \cdot x_2 + p_3 \cdot x_3 \tag{6.2}$$

Jelikož je předem dán počet přímek, se kterými se pracuje, může se násobení v implementaci nahradit podmínkou na konrétní index přímky. V pseudokódu by taková implementace mohla vypadat následovně

Po vzoru již existujících algoritmů zmíněných v předchozích kapitolách se trénují dva histogramové modely — jeden pro popředí a druhý pro pozadí. Model popředí se trénuje z uživatelské anotace a model pozadí se trénuje z úzkého pruhu kolem okraje snímku. Pixely pro trénování modelu pozadí jsou tak definovány implicitně, aby byl dbán důraz na co nejmenší potřebu uživatelského vstupu. Dolní okraj je vynechán, protože se v něm často může nacházet část objektu popředí.

Klasifikace pixelů snímku pak probíhá nezávisle na každém pixelu následujícím způsobem

Algorithm 2 Rychlá projekce bodu na přímku

```
function PROJECT_POINT_ONTO_LINE(point p, line_index i)
    if line_index = 1 then
        return p_1
     ...
    else if line_index = 5 then
        return (p_1 + p_2 − p_3 + 255)/3
     ...
    end if
end function
```

1. Vypočtou se projekce na všechny zvolené přímky a vytvoří se příslušný počet histogramů (13)
2. Zjistí se hodnoty příslušných binů v histogramu popředí a pozadí
3. Vypočte se suma rozdílů mezi váhami popředí a pozadí
4. Proběhne porovnání sumy s předem zvoleným prahem (nulou), které rozhodne o náležitosti pixelu

6.2.1 Výběr nejvhodnějších přímek

Dalším vylepšením algoritmu, které vedlo ke zrychlení a zvýšení robustnosti byl výběr pouze několika přímek dobře rozlišujících popředí od pozadí. Měření dobré rozlišitelnosti probíhalo odečtením normalizovaných histogramů pozadí od popředí. Několik přímek, jejichž histogramy měly v této metrice nejvyšší hodnoty pak bylo vybráno pro klasifikaci. Snížením počtu přímek tak dojde ke zrychlení a navíc jsou ignorovány ty projekce, jejichž histogramy pro popředí a pozadí se od sebe příliš neliší.

Je důležité zmínit, že tento výběr probíhá pokaždé, když uživatel provede anotaci na novém snímku. Pro každý objekt se tak automaticky vybere kombinace přímek, které dokáží nejlépe odlišit objekt od zbytku pozadí.

6.3 Zlepšení robustnosti rozhodovacími stromy

Pro zlepšení přesnosti algoritmu při pokusu o zachování stejné rychlosti byla naivní implementace použita jako základ pro trénování náhodných rozhodovacích stromů. Již hotový histogramový model lze totiž při trénování využít.

Abychom mohlo trénování rozhodovacích stromů proběhnout, je potřeba určit, jaká bude podmínka určující cestu dat stromem. V tomto případě půjde o stejnou podmínku jako u klasifikace v histogramovém modelu, tedy porovnání rozdílu mezi váhou popředí a váhou pozadí získanou z daného histogramu. Podmínka rozhodující cestu ve stromě pak bude rozdíl mezi dvěma obecně různými projekcemi pozadí a popředí.

Protože je implementováno dříve zmíněné rozšíření, které používá pro klasifikace několik náhodně natrénovaných stromů, je do trénování modelu potřeba zavést prvek náhody. Prvku náhody se dosáhne tím, že při výběru projekcí, které dělí vstupní množinu při největším zisku, se bude počítat zisk jen pro náhodně zvolenou podmnožinu dvojic. Navíc je povoleno používat dvě různé projekce pro popředí a pozadí.

Nejen že tato technika zavádí do modelu náhodnost, ale také výrazně sníží čas trénování stromů, protože se nemusí provádět výpočet zisku pro všechny kombinace projekcí. V kapitole 8 je pak názorně ukázáno, že tento model dosahuje větší přesnosti než histogramový model a navíc bez značných úbytků na rychlosti.

Kapitola 7

Prototyp mobilní aplikace

V této kapitole je popsán postup vytváření funkčního prototypu výsledné aplikace na vytváření fotografické montáže od prvotního návrhu uživatelského rozhraní přes jeho následnou úpravu kvůli návaznosti na vytvořené algoritmy.

Kapitola zmiňuje problematiku možnosti zpracování a vykreslování videa v reálném čase a postup integrace zmíněných algoritmů otestovaných v desktopovém prostředí do mobilní aplikace běžící na operačním systému iOS.

7.1 Návrh a implementace uživatelského rozhraní

Součástí vývojového procesu uživatelského rozhraní by měl být vždy krok, ve kterém je vytvořen základní koncept fungování aplikace. Tento koncept poté slouží jako předloha při vytváření aplikace, případně může sloužit k prezentaci funkcionality výsledné aplikace bez nutnosti psát jakýkoliv kód.

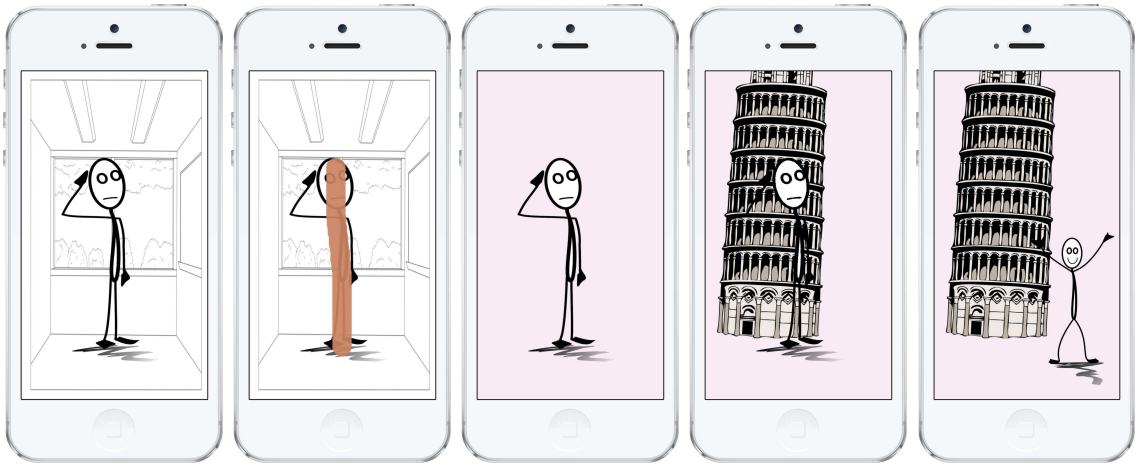
Pro aplikaci byl vytvořen základní koncept pomocí grafického programu GNU Image Manipulation Program (GIMP) a doplněn o vzhled mobilního telefonu v internetové aplikaci mockuphone [22]. Jednotlivé obrazovky konceptu jsou vyobrazeny na obrázcích 7.1. Zleva se jedná o počátek snímání uživatele kamerou, následné hrubé označení pixelů náležících do popředí, poté výběr fotky, do které chceme uživatele montovat a posledním krokem je vhodné pozicování uživatele jak v realitě, tak ve fotce.

Tento návrh byl přepracován a konkretizován do podoby odpovídající vytvořeným algoritmům do obrázku 7.2. Dále bylo potřeba zvážit umístění obrazovky s výběrem fotografie pozadí. Přestože výsledný algoritmus ve většině případů funguje obstojně, jsou situace, ve kterých se v segmentaci vyskytují chyby znemožňující vytvoření výsledné montáže. Z pohledu počítačového vidění to velký problém není, ale z uživatelského hlediska se jedná o větší překážku.

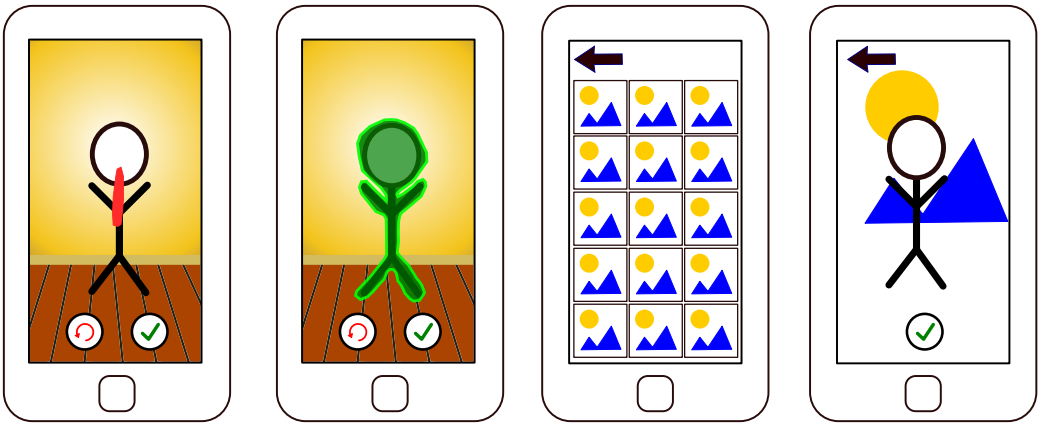
Každá aplikace se musí snažit o snížení uživatelské frustrace na minimum, tudíž byla po patřičném zvážení obrazovka s výběrem obrázku pozadí umístěna až za obrazovku s výběrem objektu popředí. To je zdůvodněno tím, že pokud se uživatel bude vyskytovat v prostředí, kde má segmentační algoritmus problémy (tj. místa tmavá nebo obecně s nízkou barevnou variabilitou), uvidí kvalitu segmentace okamžitě a tím ušetří čas se zbytečným krokem výběru.

Celý návrh se tedy skládá ze čtyř obrazovek (stavů aplikace):

- obrazovky výběru objektu popředí,



Obrázek 7.1: Koncept UI aplikace



Obrázek 7.2: (a) Uživatel vyznačí štětcem objekt popředí (b) Ukázka segmentace (c) Výběr obrázku pozadí (d) Úprava orientace pozadí

- rychlé ukázky segmentace pomocí přehledného překryvu,
- výběru fotografie pozadí a
- editační obrazovky s úpravou pozice pozadí pomocí gest.

Ideálně tedy práce s aplikací vypadá tak, že uživatel se rozhodne vytvořit fotografickou montáž, zapne aplikaci, tahem prstu vybere objekt popředí, který chce umístit do výsledné montáže, rozhodne se, jestli je segmentace dostatečně kvalitní a podle toho výběr potvrdí. Následně vybere z galerie pozadí, upraví ho pomocí zaběhlých gest pro rotaci a zoom a jako poslední krok vytvoří výslednou montáž.

Ačkoliv aplikace klade důraz na jednoduchost, v současném stavu se jedná pouze o prototyp, který není zatím připraven pro publikování do obchodu s aplikacemi. Hlavním důvodem tohoto stavu je fakt, že aplikace zavádí v současnosti neobvyklé postupy (výběr objektu popředí tahem prstu) a neobsahuje seznamovací návod.

Vytvoření aplikace vhodné pro publikování na AppStore však nebylo hlavním záměrem této práce. Pro budoucí možnost publikování takové aplikace by bylo potřeba navázat spolupráci se zručným grafikem a vytvořit přehledný a elegantní návod k použití.

7.2 Vykreslování videa v reálném čase

Ačkoliv by se zdálo, že zobrazení videa při frekvenci 30 snímků za sekundu je na mobilním zařízení běžnou a jednoduchou operací, v případě aplikačního rozhraní dostupné v operačním systému iOS tomu tak není. Hlavním problémem je vykreslování na obrazovku programátorem již upravených snímků videa. Pro jednoduché zobrazení snímků přijímaných přímo z foťáků existuje možnost použití připravené třídy `AVCapturePreviewLayer`.

Přijímaní obrázků z kamery poskytuje knihovna AVFoundation, která implementuje následující koncepty:

- Abstraktní reprezentaci nahrávacího zařízení AVCaptureDevice
- Připojení k rozhraní (kameře) mobilního telefonu přes AVCaptureSession
- Připojení vstupních a výstupních entit pomocí AVCaptureConnection
- Příjímaní RGBA snímků v entitě typu CVPixelBuffer
- Efektivní recyklaci objektů typu CVPixelBuffer pomocí CVPixelBufferPool

Po přijmutí obrázků z kamery a aplikaci algoritmu lze výstup zobrazovat v reálném čase pouze za pomoci OpenGL. Operační systém iOS poskytuje standardní rozhraní OpenGL ES 2.0, které umožňuje potřebnou konverzi obrazového bufferu do textury a následné vykreslení této textury na obrazovku.

Vykreslování videa probíhá s použitím speciálního kontextu EAGLContext reprezentujícího aktuální stav OpenGL. Obrázek videa je nejprve nakopírován do textury pomocí k tomuto účelu dostupné metody `CVOpenGLESTextureCacheCreateTextureFromImage`. Připravený vertex shader poté interpoluje texturovací souřadnice pro fragment shader a ten podle těchto souřadnic vykreslí texturu do framebufferu. Protože je výstup framebufferu napojen na speciální vykreslovací vrstvu `CAEAGLLayer`, je textura v reálném čase zobrazena na obrazovce [25].

7.3 Integrace vytvořeného algoritmu

Protože byl již původní algoritmus v desktopovém prostředí vytvářen s myšlenkou co nejjednodušší přenositelnosti a implementován v jazyce C bez použití knihoven třetích stran (OpenCV a podobné), byla integrace algoritmu velmi snadná.

Využito bylo pak především faktu, že jazyk ObjectiveC, ve kterém je mobilní aplikace implementována je rozšířením jazyka C o objektové paradigma. Jakákoliv konstrukce zapsaná v jazyce C by tedy měla být validní konstrukcí v jazyce ObjectiveC.

Pro přehlednost byla vytvořena třída `DecisionTreeWrapper`, která obaluje výsledný algoritmus a inicializuje ho s parametry, které byly při testování určeny jako nejefektivnější. Třída tak poskytuje objektové rozhraní k jinak kompletně imperativní implementaci. Navíc se třídá stará o automatickou dealokaci paměti zavoláním dealokačních metod algoritmu v destruktoru.

Ukázkový výstup z aplikace je pak vidět na obrázku 7.3. Vrchní řádek jsou uživatelské anotace provedené tahem prstu a dolní řádek jsou ukázky jednoho snímku interaktivní segmentace.

Obrázek 7.3: Pohled na výsledný algoritmus integrovaný do mobilní aplikace

7.4 Možnosti paralelizace na platformě iOS

Programátorské rozhraní dostupné na platformě iOS nabízí několik možností, jak efektivně paralelizovat sekvenční kód. Jsou to možnosti paralelizace jak na vícejádrovém procesoru, tak i na mobilních zařízeních dostupném integrovaném grafickém čipu. Pro paralelizaci na grafickém čipu pak poskytuje iOS kromě průmyslového standardu OpenGL také svoje proprietární rozhraní s názvem Metal.

7.4.1 Grand Central Dispatch

Grand Central Dispatch (GCD) je soubor vlastností jazyka, knihoven a systémových vylepšení, která poskytují možnost paralelního vykonávání programu [23]. Programátor má výhodu, že se nemusí starat o explicitní synchronizaci vláken a místo toho pracuje s takzvanými frontami (dispatch queues), na které zasílá bloky kódu. Blok kódu je speciální objekt, který reprezentuje vykonávanou práci. Bloky se značně podobají funkčním ukazatelům v jazyce C.

7.4.2 GPGPU API Metal

GCD je však pouze nástrojem paralelizace kódu na CPU. V posledních letech se rozmáhá i paralelizace na grafických čipech a mobilní telefony v tomto trendu nejsou výjimkou. Proto v září roku 2014 představila firma Apple nové programátorské rozhraní (API) pro nízkoúrovňovou práci s grafickými čipy [21]. API s názvem Metal funguje na mobilních čipech řady A7 a vyšší.

Obrázek 7.4: Ukázka aplikace na iPadu pracující s Grabcutem

Metal oproti současné implementaci OpenGL dostupné na mobilních zařízeních s operačním systémem iOS poskytuje i takzvané compute shadery, které umožňují implementaci obecných algoritmů. Navíc poskytuje Metal optimalizace, které snižují výpočetní cenu přístupu na GPU jako jsou předkompilované shadery.

Compute shadery lze využívat pro paralelizaci algoritmů počítačového vidění, jelikož často jejich části zpracovávají velkou množinu pixelů, nad kterými se vykonává stejná operace. Takové algoritmy jsou pak ideální pro implementaci na grafické kartě.

7.5 Prototyp práce s Grabcutem v iOS

V rámci této práce byl vytvořen prototyp, který ukazuje možnosti práce s algoritmem Grabcut. Algoritmus je dostupný z obecné knihovny pro počítačové vidění OpenCV. GrabCut lze používat ve dvou módech. Prvním módem je inicializace pomocí obdélníku (parametr `GC_INIT_WITH_RECT`) a druhý mód pro inicializaci pomocí masky (parametr `GC_INIT_WITH_MASK`). První mód slouží pro základní hrubou segmentaci a druhý mód pro doladění detailů explicitním označením pixelů popředí nebo pozadí.

Stejně jako algoritmus je i implementovaný prototyp rozdělen do dvou obrazovek korespondujících se zmíněnými módy. Na první obrazovce uživatel může vybrat ve statickém obrázku obdélník, ve kterém se vyskytuje popředí. Aplikace následně dá uživateli najevo, že zpracovává požadavek (samotný algoritmus je prováděn ve zvláštním vlákně) a poté mu zobrazí výsledek segmentace. Pokud uživatel není s prvotní segmentací spokojen, může přejít na další obrazovku. Na další obrazovce má možnost explicitně vyznačovat náležitost pixelů. Jestli označuje pixely popředí nebo pozadí se odvíjí od skutečnosti, jaký typ štětce si vybere. Pro snadnější práci aplikace umožňuje i volbu tloušťky štětce. Prototyp úplně neodpovídá navrženému uživatelskému rozhraní, ale je tomu tak z důvodu rychlejšího testování výstupu algoritmu přímo na zařízení. Ukázka aplikace je na obrázku 7.4.

Bohužel současná implementace algoritmu nedosahuje rychlosti potřebné pro zpracování videa. Řádově jeden snímek trvá vysegmentovat několik sekund. Z těchto důvodů bude v navazující práci nutno najít vhodný algoritmus, který bude segmentovat video v reálném čase v dostatečně kvalitním rozlišení pro realistickou fotografii.

7.6 Implementace desktopové verze Background cutu

Dalším prototypem, který byl vytvořen, je základní implementace algoritmu Background cut podle postupu zmíněného v kapitole 4.3. Prozatím je implementován pouze základ algoritmu a současná implementace neobsahuje ochranu proti otřesům kamery.

Celá aplikace je implementována jako program příkazového řádku, který dokáže zpracovávat video jak ze vstupního zařízení, tak i ze souboru. Výstupem aplikace je podle zvolených parametrů buď video s výslednou segmentací nebo zobrazení jednotlivých kroků algoritmu (vytvoření trimapy, kontrast a konečná segmentace). Celá aplikace je implementována taktéž s použitím knihovny OpenCV, ale samotný Background Cut v knihovně OpenCV není a jde tak o vlastní implementaci.

Program mimo jiné také umožňuje předtrénování modelu pozadí, což je časově nejnáročnější operace a jeho následné uložení do souboru a načtení při dalším zpracování videa. Obrázek 7.5 zobrazuje ukázkový výstup z aplikace.

Obrázek 7.5: Ukázka výstupu aplikace implementující Background cut

7.7 Urychlení GMM pomocí Metal API

Téměř všechny prozkoumané algoritmy na cutting v nějaké svojí části využívají trénování GMM. Jedním z nejznámějších algoritmů na trénování GMM je algoritmus Expectation Maximization (EM) [10]. EM je algoritmus, který iterativně hledá maximální hodnotu likelihood zadaného modelu pomocí dvou kroků — kroku expectation a kroku maximization. Některé části těchto kroků lze paralelizovat za pomocí již zmíněných compute shaderů na grafické kartě. O takové zrychlení usilovala třetí implementovaná aplikace. Určitého zrychlení se dosáhnout podařilo, ovšem stále nedosahovalo rychlosti použitelné pro online zpracování ani u obrázků s malým rozlišením.

Kapitola 8

Testování algoritmu

Pro objektivní zhodnocení je potřeba vytvořené algoritmy otestovat nad konkrétní datovou sadou. Tato kapitola se zabývá testováním jak přesnosti obou vytvořených algoritmů, tak jejich rychlostí na mobilních i desktopových zařízeních. Je zde také prezentován výstup segmentace a diskutováno vizuální zhodnocení tohoto výstupu.

Je představena datová sada, nad kterou se experimenty prováděly a také je řečeno, jak bylo potřeba datovou sadu upravit a jaké kroky vedly k její dodatečné anotaci pro kompatibilitu s vytvořenými algoritmy.

8.1 Datová sada — BSDS500

BSDS500 je zkratka pro Berkeley Segmentation Dataset and Benchmarks. Jedná se o datovou sadu, která, jak už název napovídá, obsahuje 500 obrázků s různou tématikou [8]. Původně je rozdělena do tří podmnožin — testovací, validační a trénovací sady. V případě vyhodnovaní našeho algoritmu však nemusíme toto rozdělení respektovat, protože klasifikátor se vždy trénuje na jedné konkrétní fotografii.

Datová sada obsahuje různorodé fotografie, včetně takových, na kterých není na první podle zřejmý objekt popředí (např. fotky krajiny apod.). Z tohoto důvodu byla z této datové sady vybraná pouze podmnožina fotek, na kterých je objekt popředí (člověk nebo zvíře) naprosto zřejmý. Pro testování algoritmu byla tato datová sada vybrána, protože poskytuje ke každé fotografii soubor, ve kterém je uložena ruční segmentace provedená člověkem. Každý objekt má poté svojí vlastní číselnou hodnotu a v souboru jsou vždy vyznačeny pixely, které tomuto objektu náleží příslušným číslem.

BSDS také obsahuje sadu benchmarků, nad kterými lze vyhodnotit správnost segmentace a lze tak jednoduše porovnat efektivitu algoritmu s ostatními. V případě vyvíjeného algoritmu je však řešena mírně odlišná úloha (s odlišným výstupem) než klasická segmentace a proto poskytnuté benchmarky nebyly použity.

Ukázka několika vzorků datové sady je vidět na obrázku 8.1.

8.2 Anotace

Ačkoliv by šla anotace vzorků vytvořit i v generickém rastrovém grafickém editoru jako je například GNU Image Manipulation Program (gimp), pro větší pohodlnost a rychlost anotace byl vytvořen speciální anotační nástroj. Zmíněný nástroj byl realizován jako program příkazové řádky, který je vystavěn nad grafickým rozhraním dostupným v knihovně

Obrázek 8.1: Ukázka datové sady BSDS500

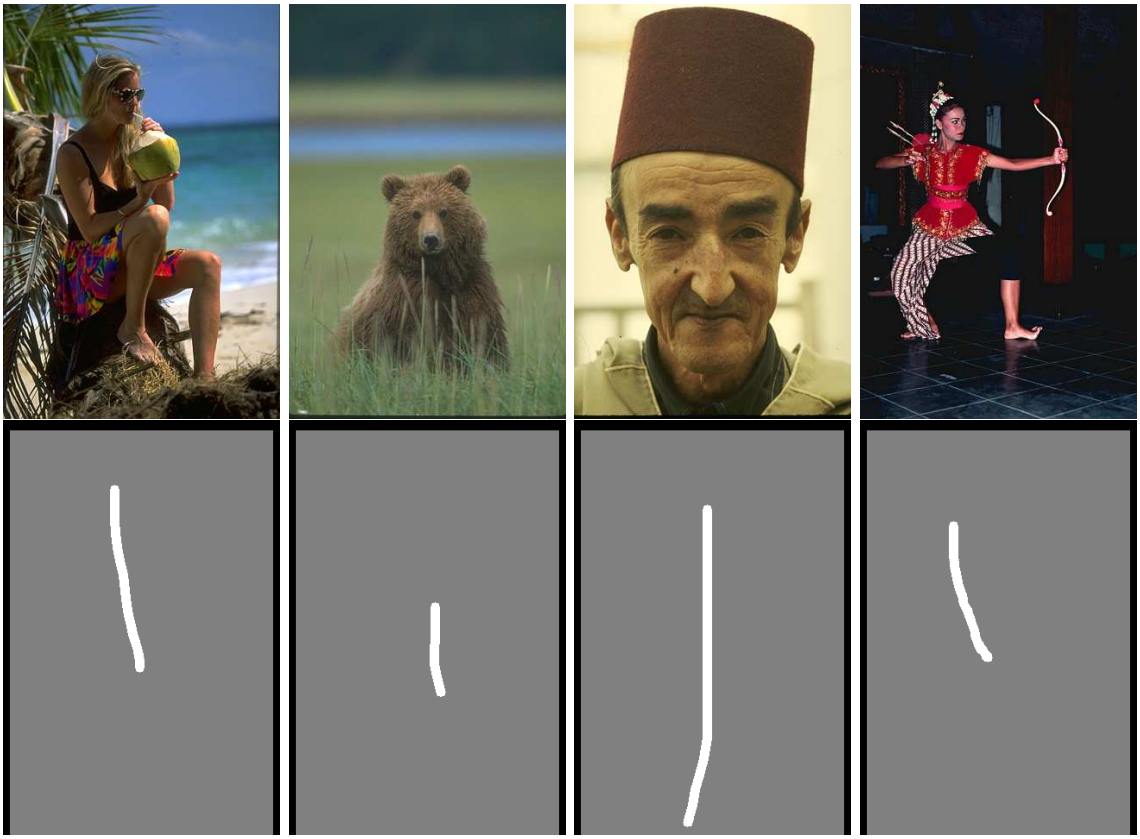
OpenCV.

Anotace vzorků probíhá ve dvou krocích. Nejprve se automaticky označí úzký pruh po celém okraji obrázku s výjimkou spodního okraje a označí se jako pozadí (černá barva), a poté uživatel vyznačí tahem myši, kde se nachází objekt popředí. Spodní okraj je vynechán, protože se při náhodném testování aplikace objekty zájmu často v této oblasti vyskytovaly. Pokud by spodní okraj byl označen jako pozadí, mohlo by to vést k celkové degradaci efektivity algoritmu.

Po interaktivním označení popředí může uživatel potvrdit výběr libovolnou klávesou a výsledná maska je uložena do souboru, který byl předán jako parametr aplikace na příkazovém řádku. Ukázka obrázků a jejich vytvořených anotací je na obrázku 8.2

V rámci této práce byl anotační nástroj použit pro vyznačení objektu popředí na vybrané podmnožině vzorků datové sady BSDS500. K tomuto účelu byl vytvořen krátký skript, který spouští anotátor nad každým obrázkem zvoleného adresáře a ukládá masky ve formátu PNG do libovolného adresáře.

Celkem bylo vybráno a označeno 213 obrázků, které odpovídaly scénářům možných použití vytvářené aplikace a nad nimiž byla následně vyhodnocena přesnost algoritmů popsána v navazujících kapitolách.



Obrázek 8.2: Obrázky s ukázkou jejich ruční anotace

8.3 Metodika testování

Nad vybranou datovou sadou je třeba zvolit vhodnou metodiku testování vytvořených algoritmů. Je potřeba uvést, že binární klasifikátor našeho typu může vracet 4 různé typy výsledků, kterými jsou:

- True positive (TP) — klasifikátor správně označil vzorek popředí jako popředí
- True negative (TN) — klasifikátor správně označil vzorek pozadí jako pozadí
- False positive (FP) — klasifikátor nesprávně označil vzorek pozadí jako popředí
- False negative (FN) — klasifikátor nesprávně označil vzorek popředí jako pozadí

můžeme tedy narazit na dva typy úspěšné klasifikace a dva typy chyb. Bez této znalosti by pravděpodobně byla zvolena metriku, kterou lze popsat rovnicí

$$accuracy = \frac{CORRECT}{N}$$

tato metrika však kompletně opomíjí rozdělení chyb i úspěchů na dva druhy.

Proto se zavádí jiné vyhodnocovací metriky, mezi které patří i v této práci použitý precision a recall. Precision je definován jako

$$precision = \frac{TP}{TP + FP}$$

a recall jako

$$recall = \frac{TP}{TP + TN}$$

Dohromady se pomocí těchto dvou metrik tvoří precision-recall křivka, která je nepsaným standardem ve vyhodnocování úspěšnosti binárních klasifikátorů. Dokáže totiž ukázat, jak se binární klasifikátor chová při změně jeho prahu a zároveň ukázat, jaké hodnoty prahu můžeme nastavit, pokud chceme dosáhnout konkrétní chybovosti.

Pro porovnání byla testována jak naivní implementace, tak vylepšený model využívající náhodných rozhodovacích stromů. Přesnost algoritmu byla z důvodu výkonu testována pouze v desktopové aplikaci, kdežto rychlost byla vyhodnocena jak pro desktopovou, tak pro mobilní verzi přímo ve vytvořené aplikaci.

8.4 Přesnost

Pro testování přesnosti byly zvoleny následující parametry modelu:

- počet binů histogramu pro naivní klasifikátor (8, 16, 32, 64, 128, 256, 512 a 1024),
- výška rozhodovacího stromu od 2 do 5,
- počet binů histogramu použitých pro trénování rozhodovacích stromů (8, 64, 256, 1024) a
- počet rozhodovacích stromů (2, 4, 8 a 10).

U naivního histogramového modelu byl testován pouze vliv počtu binů na kvalitu výsledné segmentace. Histogramový model totiž nemá jiné parametry (kromě dále zmíněného prahu). Pro sestrojení precision-recall křivek byl rozhodovací práh nastavován na 30 hodnot, rovnoměrně rozložených mezi v intervalu $< -3, 3 >$. Výsledná křivka byla poté sestrojena lineární interpolací mezi datovými body. Výsledky byly vyneseny do grafu na obrázku 8.3 . Z grafu je patrné, že si nejlépe vedl klasfikátor s 256 biny. Z toho důvodu byla tato hodnota nastavena na pevno do výsledné aplikace.

Pro rozhodovací stromy je výpočet PR křivky obtížnější, protože se při klasifikaci používá celočíselný práh. Nejprve tak byl proveden výpočet pro všechny kombinace parametrů pro celočíselný práh nastavený na 0. Prezentované hodnoty precision a recall jsou průměrem nad celou datovou sadou. Výsledek byl jednak vynesen do grafu 8.4 a pro přehlednost doplněn tabulkami 8.1, 8.2, 8.3 a 8.4 s parametry seskupenými podle výšky stromu.

Nejlepší precision při prahu 0 měl nad datovou sadou tedy klasifikátor s konfigurací 1024 binů histogramu, při hloubce stromu 4 a počtu stromů také 4.

8.5 Rychlost

Pro vysokou interaktivitu implementované aplikace je potřeba, aby algoritmy provádějící segmentaci pracovaly v reálném čase. V této kapitole je vyhodnocena rychlost běhu naivního histogramového klasifikátoru pro různá rozlišení dostupná na mobilní platformě.

Programátorské rozhraní operačního systému iOS umožňuje přijímat video v rozlišeních specifických pro konkrétní zařízení nebo pro předdefinované hodnoty 352x288, 640x480 a 1280x720 pixelů. Jiná rozlišení stejná na vsěch zařízeních nativně poskytována nejsou.

	Počet stromů			
Počet binů	2	4	8	10
8	56%/34%	59%/38%	58%/54%	58%/57%
16	62%/44%	63%/46%	60%/60%	61%/58%
64	65%/48%	65%/49%	62%/62%	63%/59%
256	64%/46%	66%/49%	63%/61%	62%/61%
1024	65%/46%	66%/47%	62%/61%	62%/61%

Tabulka 8.1: Precision/Recall pro výšku stromu 2

	Počet stromů			
Počet binů	2	4	8	10
8	60%/49%	61%/54%	58%/62%	58%/61%
16	63%/51%	65%/56%	61%/66%	61%/64%
64	67%/55%	67%/60%	63%/68%	63%/67%
256	67%/55%	66%/59%	63%/68%	64%/67%
1024	66%/53%	66%/59%	63%/67%	64%/66%

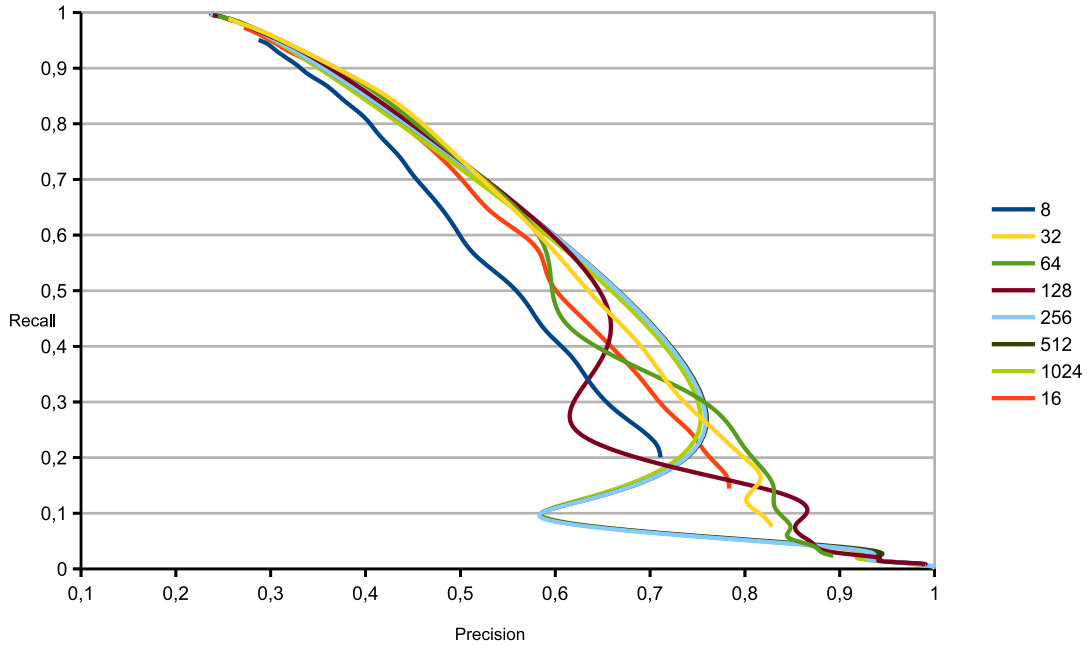
Tabulka 8.2: Precision/Recall pro výšku stromu 3

	Počet stromů			
Počet binů	2	4	8	10
8	63%/44%	63%/49%	60%/58%	60%/60%
16	65%/50%	67%/54%	63%/64%	64%/62%
64	70%/50%	69%/56%	65%/66%	66%/64%
256	69%/51%	69%/57%	66%/65%	66%/63%
1024	70%/50%	70%/54%	65%/64%	66%/63%

Tabulka 8.3: Precision/Recall pro výšku stromu 4

	Počet stromů			
Počet binů	2	4	8	10
8	62%/35%	64%/38%	60%/55%	62%/53%
16	65%/42%	65%/46%	63%/59%	64%/57%
64	69%/45%	70%/51%	66%/61%	66%/61%
256	69%/46%	69%/50%	66%/61%	67%/59%
1024	69%/42%	69%/48%	66%/58%	67%/59%

Tabulka 8.4: Precision/Recall pro výšku stromu 5



Obrázek 8.3: Precision-Recall křivky pro různé počty binů histogramového modelu

Testování rychlosti bylo rozděleno na fázi učení, kdy se z jednoho snímku a uživatelské anotace natrénuje a fázi klasifikace jednoho snímku. Pro získání co nejobjektivnějších výsledků byla rychlost získána jako průměrná hodnota z několika nezávislých běhů.

Tabulka 8.6 nabízí přehled výsledků běhu trénování a klasifikace na jednom snímku na desktopu při různých rozlišeních. Každá hodnota je vždy průměrem z několika běhů. Také tabulka 8.5 poskytuje stejné měření pro desktopovou implementace, ovšem nyní za použití stromového klasifikátoru.

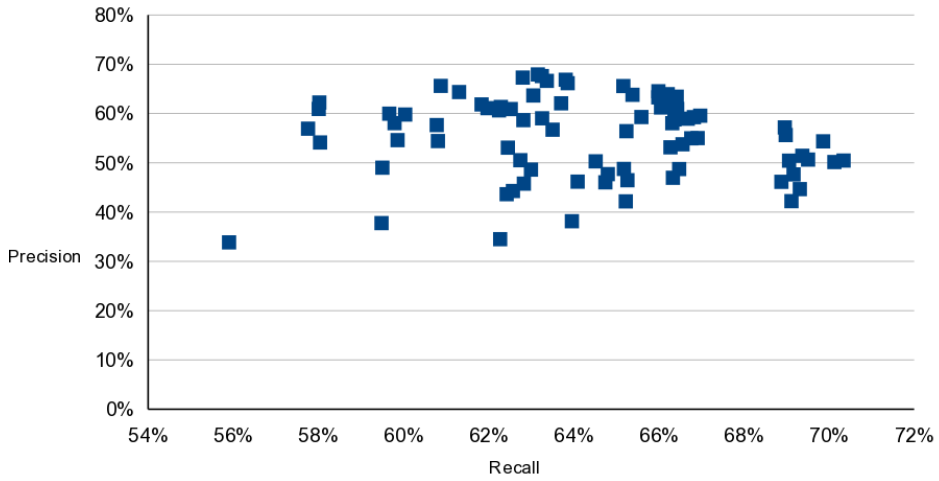
Rozlišení	Čas trénování	Zpracování snímku
352x288	0.4 s	0.15 s
640x480	1.1 s	0.048 s
1280x720	2.1 s	0.1 s

Tabulka 8.5: Měření času modelu s rozhodovacími stromy v desktopovém prostředí

Tabulky 8.7 a 8.8 poté zapisují provedená měření na mobilním telefonu pro histogramový model, respektive pro model s rozhodovacími stromy. Všechna měření byla prováděna pro výšku stromu 3, počet 4 stromů a 256 binů histogramu.

Jak je vidět, algoritmus sice neběží rychlostí 30 snímků za sekundu, ale pro základní interaktivitu jsou tyto časy dostačující. Vyšší rozlišení ale momentálně nelze použít, protože by se interaktivita aplikace snížila na minimum díky dlouhému výpočtu jednoho snímku i úvodního trénování. Oproti v úvodu zmíněným algoritmům, které běžely i na nízkých rozlišeních v řádu několika sekund je to určitě zlepšení.

Ačkoliv je klasifikátor založený pouze na histogramovém modelu méně robustní, jeho



Obrázek 8.4: Precision a recall pro všechny kombinace parametrů rozhodovacích stromů

Rozlišení	Čas trénování	Zpracování snímku
352x288	0.002 s	0.022 s
640x480	0.005 s	0.05 s
1280x720	0.01 s	0.012 s

Tabulka 8.6: Měření času histogramového modelu v desktopovém prostředí

ukazuje se jeho nesporná výhoda v rychlosti natrénování. Při použití histogramového klasifikátoru nemusí být uživateli ani prezentován indikátor činnosti (známá "loading"obrazovka).

Dále je z výsledků vidět, že mobilní telefon je pouze zhruba 6x pomalejší než počítač, na kterém bylo testování prováděno (Macbook air s procesorem i5).

8.6 Vliv přesnosti uživatelské anotace

Velmi důležitou vlastností algoritmu, který má být uživatelsky přívětivý, je robustnost vůči nepřesnostem v uživatelském označení objektu popředí. Jelikož je i cílem tohoto projektu vytvoření aplikace, kterou by mohl používat běžný uživatel, byla zmíněná vlastnost testována.

Na obrázku 8.5 je ukázka vybraných obrázků s několika anotacemi v různé přesnosti. Aby bylo testování co nejobjektivnější, byly pro všechny obrázky zvoleny jednotné parametry. Tato volba navíc odpovídá i reálnému nastavení výsledné aplikace, ve které jsou

Rozlišení	Čas trénování	Zpracování snímku
352x288	0.01 s	0.07 s
640x480	0.03 s	0.12 s
1280x720	0.06 s	0.28 s

Tabulka 8.7: Měření času histogramového modelu na mobilním telefonu

Rozlišení	Čas trénování	Zpracování snímku
352x288	3 s	0.15 s
640x480	7 s	0.33 s
1280x720	15s	1.37 s

Tabulka 8.8: Měření času modelu s rozhodovacími stromy na mobilním telefonu

parametry nastaveny napevno pro dosažení ideálního poměru přesnosti a rychlosti.

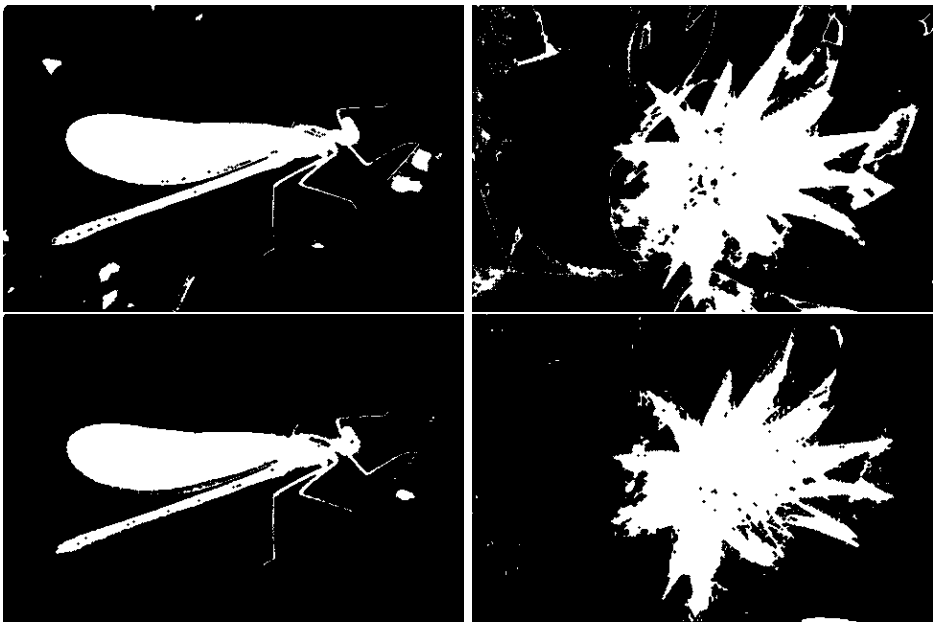
Z obrázku je patrné, že model je mírně citlivý na přesnost anotace. Pokud uživatel zasáhne kousek pozadí, může se stát, že ve výsledné segmentaci bude více šumu, jako je tomu na zmíněném obrázku.

8.7 Porovnání výsledných segmentací

Ačkoliv jsou výše zmíněné metriky velmi objektivní, je také vhodné ukázat výsledky segmentace v obrázcích. V obrázcích 8.6 a 8.7 je ukázáno, jak se vylepšením modelu rozhodovacími stromy viditelně zlepšila segmentace. Masky v prvním řádku jsou vždy segmentované pomocí histogramového modelu a ve druhém řádku jsou segmentace pomocí rozhodovacích stromů.

I opticky jsou mezi jednotlivými obrázky znatelné rozdíly, zejména pak co se týče okolního šumu kolem segmentovaného objektu. Klasifikátor s rozhodovacími tak znovu ukazuje, že je robustnějším řešením ve srovnání s pouhým histogramovým modelem.

Obrázek 8.5: Chyby při nepřesné anotaci



Obrázek 8.6: Porovnání segmentace histogramovým modelem a modelem s rozhodovacími stromy

Obrázek 8.7: Porovnání segmentace histogramovým modelem a modelem s rozhodovacími stromy

Kapitola 9

Navazující práce

Algoritmus v aplikaci je dostačující pro většinu běžných scénářů. Na rychlosti a přesnosti však lze stále pracovat. Zatím nebylo využito možností paralelizace na platformě iOS u multijádrových procesorů. V tomto směru by mohl pokračovat další vývoj algoritmu.

Do výsledné aplikace by také bylo vhodné implementovat další vylepšení, jako například uživatelské filtry nebo sdílení výsledných obrázků přes sociální sítě. Grafická úprava aplikace je zatím ve fázi prototypu. Již bylo řečeno, že na finální verzi aplikace, včetně vestavěného uživatelského průvodce, by se měl podílet zručný grafik.

Po vylepšení grafické úpravy a implementaci průvodce by mělo následovat otestování uživateli. Pokud aplikace bude připravena k použití, bude vhodné jí publikovat na AppStore a sbírat uživatelskou zpětnou vazbu. Poté lze implementovat nejvíce se opakující požadavky.

Momentálně byla aplikace implementována a optimalizována pouze pro mobilní telefon iPhone 5. Vhodné by bylo prozkoumat funkcionalitu na ostatních zařízeních a provést nezbytné úpravy.

Kapitola 10

Závěr

Cílem tohoto projektu bylo prozkoumat možnosti implementace rychlé segmentace na mobilním telefonu a použít ji při vytváření aplikace na interaktivní montáž. Byl vytvořen přehled algoritmů, které jsou momentální špičkou v oboru a diskutovány jejich nedostatky z hlediska zpracování v reálném čase.

Jako odpověď na tyto nedostatky byl navržen algoritmus využívající klasifikace pomocí barevných histogramů se zaměřením na zpracování v reálném čase na mobilním telefonu. Tento algoritmus dále prošel vývojem a jeho robustnost byla vylepšena metodou náhodných rozhodovacích stromů. Obě verze algoritmu byly vystaveny objektivnímu testování a bylo ukázáno, že pro dostatečně malá rozlišení jsou schopny ve většině scénářů pracovat v reálném čase i na mobilním telefonu a to při dostatečně kvalitní segmentaci.

Bylo navrženo a implementováno uživatelské rozhraní, které umožňuje pohodlně pracovat s vyvinutými algoritmy na mobilním telefonu. Dbáno bylo na co nejmenší uživatelskou frustraci v případě, že se segmentace nezdaří díky nevhodnému prostředí. Aplikace byla vyvinuta na operačním systému iOS.

Součástí práce bylo i vytvoření několika prototypů, které sice nebyly přímo využity ve vytvořené aplikaci, ale byly cennými a inspirativními kroky při vytváření výsledné aplikace. Jmenovitě to byl prototyp práce s algoritmem Background Cut a také mobilní aplikace pro práci s algoritmem GrabCut, jejíž uživatelské rozhraní sloužilo jako základ pro hotovou aplikaci. Kromě toho je k oběma vyvinutým algoritmům dostupné desktopové rozhraní postavené na knihovně OpenCV.

Literatura

[1] Berman, A.; Vlahos, P.; Dadourian, A.: Method for removing from an image the background surrounding a selected subject by generating candidate mattes. Září 11 2001, uS Patent 6,288,703.
URL http://www.google.com/patents/US6288703

[2] Boykov, Y.; Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 26, č. 9, 2004: s. 1124–1137.

[3] Boykov, Y. Y.; Jolly, M.-P.: Interactive graph cuts for optimal boundary & region segmentation of objects in ND images. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, ročník 1, IEEE, 2001, s. 105–112.

[4] Chuang, Y.-Y.; Curless, B.; Salesin, D. H.; aj.: A bayesian approach to digital matting. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, ročník 2, IEEE, 2001, s. II–264.

[5] Jamriska, O.; Sykora, D.; Hornung, A.: Cache-efficient graph cuts on structured grids. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, IEEE, 2012, s. 3673–3680.

[6] Levin, A.; Lischinski, D.; Weiss, Y.: A closed-form solution to natural image matting. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, ročník 30, č. 2, 2008: s. 228–242.

[7] Liu, J.; Sun, J.: Parallel graph-cuts by adaptive bottom-up merging. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, IEEE, 2010, s. 2181–2188.

[8] Martin, D.; Fowlkes, C.; Tal, D.; aj.: A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics. In *Proc. 8th Int'l Conf. Computer Vision*, ročník 2, July 2001, s. 416–423.

[9] Milgram, P.; Kishino, F.: A taxonomy of mixed reality visual displays. *IEICE TRANSACTIONS on Information and Systems*, ročník 77, č. 12, 1994: s. 1321–1329.

[10] Moon, T. K.: The expectation-maximization algorithm. *Signal processing magazine, IEEE*, ročník 13, č. 6, 1996: s. 47–60.

[11] Olshen, L.; Stone, C. J.; aj.: Classification and regression trees. *Wadsworth International Group*, ročník 93, č. 99, 1984: str. 101.

[12] Rother, C.; Kolmogorov, V.; Blake, A.: Grabcut: Interactive foreground extraction using iterated graph cuts. In *ACM Transactions on Graphics (TOG)*, ročník 23, ACM, 2004, s. 309–314.

[13] Sun, J.; Jia, J.; Tang, C.-K.; aj.: Poisson matting. In *ACM Transactions on Graphics (ToG)*, ročník 23, ACM, 2004, s. 315–321.

[14] Sun, J.; Zhang, W.; Tang, X.; aj.: Background cut. In *Computer Vision–ECCV 2006*, Springer, 2006, s. 628–641.

[15] Wang, J.; Bhat, P.; Colburn, R. A.; aj.: Interactive video cutout. *ACM Transactions on Graphics (TOG)*, ročník 24, č. 3, 2005: s. 585–594.

[16] Wei, Y.; Sun, J.; Tang, X.; aj.: Interactive offline tracking for color objects. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, IEEE, 2007, s. 1–8.

[17] WWW Stránky: App Store Review Guidelines. `https://developer.apple.com/app-store/review/guidelines/`, cit. [2015-01-13].

[18] WWW stránky: Core Graphics Framework Reference. `https://developer.apple.com/library/ios/documentation/CoreGraphics/Reference/CoreGraphics_Framework/_index.html`, cit. [2015-01-13].

[19] WWW Stránky: Core Image Programming Guide. `https://developer.apple.com/library/ios/documentation/GraphicsImaging/Conceptual/CoreImaging/ci_intro/ci_intro.html`, cit. [2015-01-13].

[20] WWW Stránky: iOS Human Interface Guidelines. `https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html`, cit. [2015-01-13].

[21] WWW Stránky: Metal for developers. `https://developer.apple.com/metal/`, cit. [2015-01-13].

[22] WWW Stránky: Mockuphone. `http://mockuphone.com/`, cit. [2015-01-13].

[23] WWW Stránky: Grand Central Dispatch (GCD) Reference. `https://developer.apple.com/library/ios/documentation/Performance/Reference/GCD_libdispatch_Ref/`, cit. [2015-05-24].

[24] WWW Stránky: GridCut. `http://gridcut.com/`, cit. [2015-05-24].

[25] WWW Stránky: OpenGL ES Programming Guide for iOS. `https://developer.apple.com/library/ios/documentation/3DDrawing/Conceptual/OpenGLES_ProgrammingGuide/`, cit. [2015-05-24].