ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra kybernetiky

Automatická tvorba her typu puzzle na dotykovém zařízení s OS Android

Diplomová práce

Studijní program: Otevřená informatika (magisterský)
Studijní obor: Počítačové vidění a digitální obraz

Vedoucí práce: RNDr. Daniel Průša, Ph.D.

Bc. Václav Pruner

Praha 2015

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Václav P r u n e r

Studijní program: Otevřená informatika (magisterský)

Obor: Počítačové vidění a digitální obraz

Název tématu: Automatická tvorba her typu puzzle na dotykovém zařízení s OS Android

Pokyny pro vypracování:

Cílem práce je navrhnout aplikaci pro dotykové zařízení s OS Android, která bude podporovat sestavování skládačky z dílků. Návrh konkrétní podoby skládačky je součástí práce. Kritériem je pouze zábavnost hry, nemusí se jednat o žádnou z tradičních variant. Jako cílová skupina se předpokládá dítě ve věku 3-6 let. Významnou funkcionalitou bude automatická tvorba zadání podle zvoleného obrázku/fotografie. Na základě segmentace obrazu a dalšího zpracování budou detekovány signifikantní objekty, od kterých se bude odvíjet rozložení na dílky. Implementaci segmentace provede autor vlastní.
Úspěšnost a spolehlivost generování zadání bude analyzována pro různé typy vstupů. Kromě implementace bude vytvořena uživatelská příručka a programátorská dokumentace.

Seznam odborné literatury:

[1] Šonka M., Hlaváč V., Boyle R.: Image Processing, Analysis and Machine vision, 3rd edition, Thomson Learning, Toronto, Canada, 2007.
[2] Nudelman G.: Android Design Patterns, 1st edition, Wiley, Indianapolis, USA, 2013.

Vedoucí diplomové práce: RNDr. Daniel Průša, Ph.D.

Platnost zadání: do konce letního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 21. 1. 2015

Poděkování

Na tomto místě bych chtěl poděkovat panu RNDr. Danielu Průšovi, Ph.D. za věcné připomínky a odborný dohled nad touto prací. Dále bych chtěl poděkovat rodině za podporu během celého studia.

Prohlášení autora práce

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne ....

Podpis autora práce

Anotace

Cílem této práce je vyvinutí aplikace pro zařízení s operačním systémem Android, která automaticky vytvoří hru typu puzzle pro libovolný vstupní obrázek. Jednotlivé dílky skládanky by měly reflektovat objekty v příslušném vstupu. Automatická tvorba je založena na segmentaci obrazu a kompaktnosti geometrických útvarů. Cílovou skupinou jsou děti přibližně ve věku od tří do šesti let.

Klíčová slova

segmentace obrazu, kompaktnost geometrických útvarů, Android

Abstract

The goal of this thesis is to develop an application for devices with Android operating system, which automatically creates a puzzle game for arbitrary input image. Every piece of the puzzle should reflect objects in its respective input. Automatic creation is based on image segmentation and compactness of geometric shapes. The target group are children from about three to six years old.

Keywords

image segmentation, geometric shape compactness, Android

Obsah

Seznam obrázků

Kapitola 1

Úvod

Chytré telefony a tablety si v současné době užívají značné popularity, která, jak se zdá, bude dále jen růst. S tím je spojen i vzrůstající zájem o vývoj aplikací pro tato zařízení, který byl i pohnutkou pro výběr tématu - aplikování velmi často výpočetně náročných metod počítačového vidění na mobilních zařízeních, která mají menší výpočetní výkon a paměťové možnosti než počítače.
Cílem práce bylo navrhnout aplikaci pro zařízení s operačním systémem Android, která pro libovolný vstupní obrázek vytvoří hru typu puzzle; cílovou skupinou jsou děti ve věku od tří do šesti let. Jednotlivé dílky, nalezené ve vstupu metodami počítačového vidění, by měly reflektovat koherentní objekty v obraze. Kompletní seznam základních požadavků na vyvíjenou aplikaci je následující:

- Uživatel má možnost vybrat libovolný vstupní obrázek
- Uživatel má možnost uložit stávající skládanku
- Uživatel má možnost načíst dříve uloženou skládanku

Vlastní hraní hry probíhá „vlepováním“ vytvořených dílků zpět do původního obrázku operací „uchopit a táhnout“ („Drag & Drop“). Při vývoji byl kladen důraz zejména na návrh algoritmu, který výslednou skládanku vytvoří.

Kapitola 2

Současný stav

Aplikace pro operační systém Android jsou distribuované službou Google Play[1], konkrétně její sekcí Google Play Store (existují ještě sekce Google Play Music pro distribuci hudby, Google Play Movies & TV pro distribuci videí a Google Play Books pro distribuci elektronických knih[2]). Služba Google Play je dostupná z každého zařízení vybaveného operačním systémem Android.

Google Play obsahuje velké množství her typu puzzle pro děti, které jsou ovšem pouze variacemi na dvě varianty. První variantou jsou aplikace na základě klasických fyzických puzzlů, jedná se tedy o pouhé rozřezání obrázku. Příkladem může být aplikace Animal Jigsaw Puzzles For Kids[3], viz Obr.1.

Obr. 1 Animal Jigsaw Puzzles For Kids

Druhá varianta se principiálně přibližuje požadavkům na zadání této práce - jedná se o „vlepování“ vyřízlých objektů zpět do původního obrázku, tvorba výřezů ovšem neprobíhá automaticky a dvojice obrázek - výřezy jsou dodány vývojářem aplikace.

Obr. 2 Toddlers Puzzle Woozzle

Zástupcem tohoto typu skládanek je například hra Toddlers Puzzle Woozzle[4], viz Obr.2.
Aplikace, která byla vyvíjena jako cíl této práce, tedy nemá v distribuční službě Google Play zastoupení.

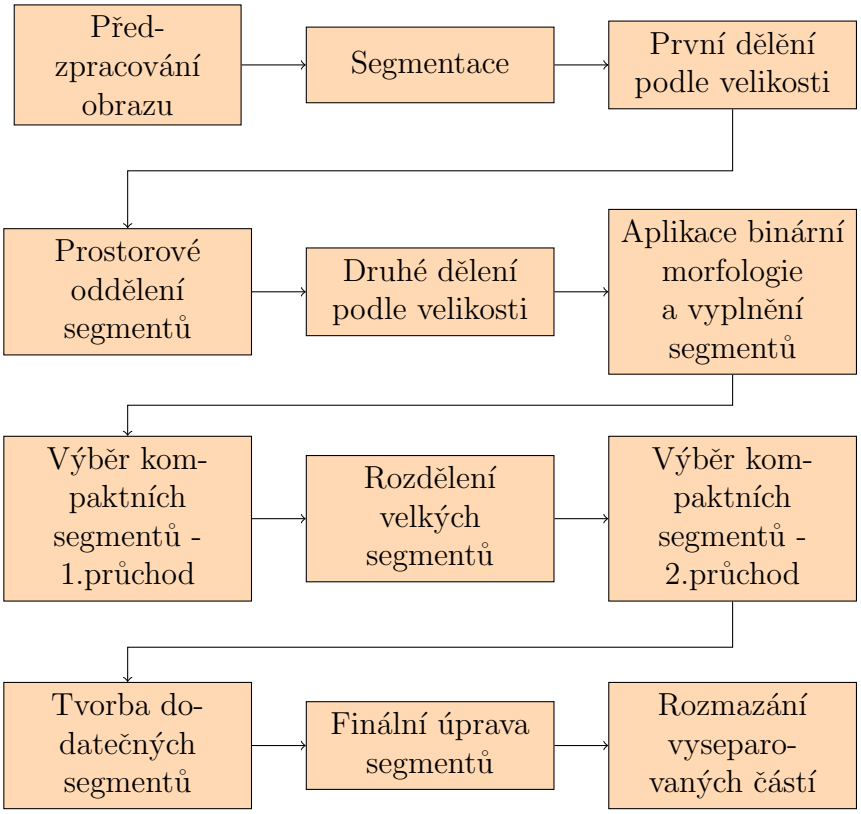
Kapitola 3

Algoritmus

Vstupem použitého algoritmu (viz. vývojový diagram Obr.3) je vybraný obrázek, přesněji pole s RGB hodnotami jeho pixelů. Výstupem jsou indexy vyseparovaných obrazců (každý obrazec má seznam indexů svých pixelů) a obrázek, který je na vyseparovaných pozicích rozmazán (opět jde tedy o pole s RGB hodnotami jeho pixelů).
Vstupní obrázek je nejprve zmenšen kvůli urychlení výpočetních časů, poté je provedena segmentace a prostorové oddělení vzniknuvších segmentů. U segmentů správné velikosti (tzn. ani příliš malých ani příliš velkých) je poté porovnána jejich kompaktnost s předem stanoveným kompaktnostním prahem. Přijaté segmenty jsou nakonec zvětšeny zpátky na původní velikost vstupního obrázku, který je na jejich pozicích rozmazán.

3.1 Předzpracování obrazu

Aby algoritmus pracoval v přijatelných časových relacích je třeba vstupní obraz nejprve zmenšit. Na jeden snímek je alokováno přibližně 250 tisíc pixelů. Obraz s poměrem stran 4:3 bude mít po takovémto zmenšení rozměry přibližně 580px x 430px, obraz s poměrem stran 16:9 přibližně 680px x 380px a obraz s poměrem stran 16:10 přibližně 640px x 400px.
Alokování menšího počtu pixelů vede sice k rychlejším výpočetním časům, dochází nicméně ke zkreslení objektů ve scéně (zejména jejich hranic), viz. Obr.4. Analogicky větší počet pixelů vede k delšímu trvání výpočtu a k lepšímu zachování tvarů.
Jako algoritmy ke zmenšení velikosti obrazu byly zvažovány interpolace pomocí nejbližšího souseda, bilineární interpolace a bikubická interpolace.



Obr. 3 Vývojový diagram algoritmu

(a) rozměry 633px x 475px **(b)** rozměry 200px x 150px

Obr. 4 Vliv přiděleného počtu pixelů na zachování tvarů v obraze

3.1.1 Interpolace pomocí nejbližšího souseda

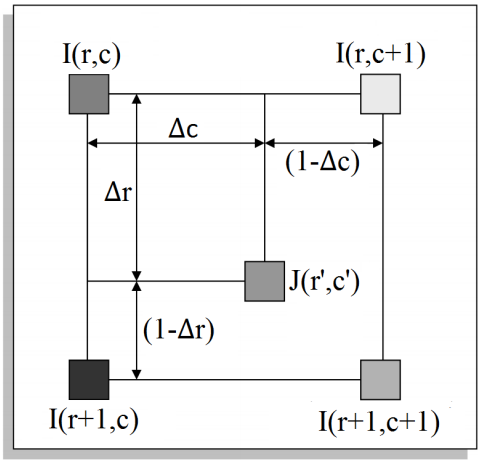
Jak již název napovídá, interpolace pomocí nejbližšího souseda (Nearest neighbour interpolation)[5][7] přiřadí na interpolovanou pozici hodnotu intenzity nejbližšího pixelu. Nevýhoda, této jinak přímočaré metody, je tvorba „schodů“ u objektů s ostrými hranicemi (patrnější u zvětšování obrazu).

3.1.2 Bilineární interpolace

Bilineární interpolace (Bilinear interpolation)[5][6] předpokládá, že funkce intenzity je lineární ve svém okolí a hodnotu intenzity interpolovaného pixelu počítá jako vážený průměr čtyřech okolních pixelů z původního obrazu.
Konkrétně je hodnota intenzity interpolovaného pixelu $J(r', c')$ vypočítána podle vzorce

$$\begin{aligned} J(r', c') = & I(r, c) \cdot (1 - \triangle r) \cdot (1 - \triangle c) \\ & + I(r + 1, c) \cdot \triangle r \cdot (1 - \triangle c) \\ & + I(r, c + 1) \cdot (1 - \triangle r) \cdot \triangle c \\ & + I(r + 1, c + 1) \cdot \triangle r \cdot \triangle c \end{aligned}$$

kde *I(x,y)* udává hodnotu intenzity původního pixelu na souřadnicích (x, y), význam ostatních proměnných je patrný z Obr.5. Bilineární interpolace může



Obr. 5 Bilineární interpolace - hodnota nového pixelu

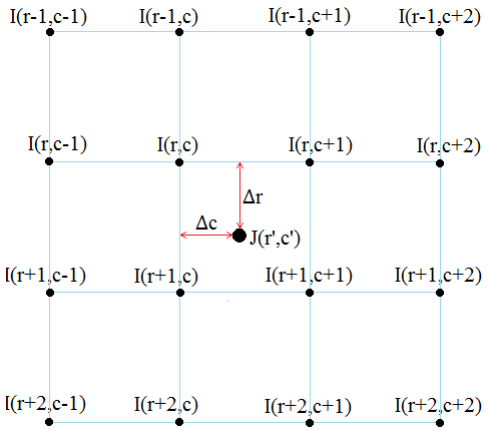
díky povaze průměrování způsobit rozmazání, redukuje efekt "schodů"představený u interpolace pomocí nejbližšího souseda.

3.1.3 Bikubická interpolace

Bikubická interpolace[5][7][8] ještě dále zpřesňuje hodnotu intenzity interpolovaného pixelu tím, že bere v úvahu šestnáct sousedních pixelů z původního obrazu (viz. Obr.6). Obecně se hodnota intenzity interpolovaného pixelu *J(r',c')* vypočte podle vzorce

$$J(r', c') = \sum_{m=-1}^{2} \sum_{n=-1}^{2} I(r + m, c + n) \cdot R_c(m - \triangle r) \cdot R_c(\triangle c - n)$$

Obdobně jako u bilineární interpolace udává *I(x,y)* hodnotu intenzity pixelu na souřadnicích *(x,y)* v původním obrazu, význam $\triangle r$ a $\triangle c$ je možno odečíst z Obr.6; funkce R_c (interpolační funkce či interpolační jádro) udává váhy intenzit každého z šestnácti sousedních pixelů původního obrazu ve výsledné sumě, z čehož plyne, že volbou vhodné funkce R_c lze vyjádřit i bilineární interpolaci a interpolaci pomocí nejbližšího souseda.



Obr. 6 Bikubická interpolace - hodnota nového pixelu

Jednou z často používaných funkcí je funkce typu Bell[7][8], viz. Obr.7(a) (i když se v pravém slova smyslu nejedná o bikubickou interpolaci, jelikož

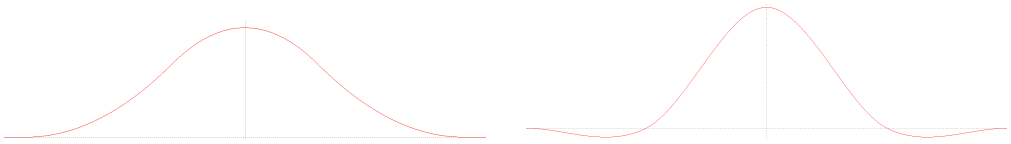
předpis neobsahuje žádnou mocninu tří)

$$R_c(x) = \begin{cases} \frac{1}{2}\left(x + \frac{3}{2}\right)^2 & -\frac{3}{2} \leq x \leq -\frac{1}{2} \\ \frac{3}{4} - x^2 & -\frac{1}{2} \leq x \leq \frac{1}{2} \\ \frac{1}{2}\left(x - \frac{3}{2}\right)^2 & \frac{1}{2} \leq x \leq -\frac{3}{2} \\ 0 & \text{jinak} \end{cases}$$

Další často používaným jádrem je Jádro CatMull-Rom[7][9], viz. Obr.7(b)

$$R_c(x) = \begin{cases} 9|x|^3 - 15|x|^2 + 6 & |x| < 1 \\ -3|x|^3 + 15|x|^2 - 24|x| + 12 & 1 \leq |x| < 2 \\ 0 & \text{jinak} \end{cases}$$

Bikubická interpolace se dokáže vypořádat s efektem "schodů"interpolace



(a) Bell **(b)** CatMull-Rom

Obr. 7 Interpolační jádra

pomocí nejbližšího souseda a potlačuje i rozmazání přítomné u bilineární interpolace. Na první pohled je však patrné, že je časově náročnější než předešlé dva způsoby.

3.1.4 Porovnání metod

Porovnání výše zmíněných čtyřech metod je demonstrováno na Obr.8. Původní obrázek Obr.8(a) o velikosti 3392px x 3328px byl zmenšen pomocí každé z výše zmíněných metod na rozměry 500px x 428px. Na snímcích Obr.8(b) až Obr.8(e) je detail (zvýrazněný zeleným obdélníkem na Obr.8(a)) každé této zmenšeniny. Je vidět, že s pokročilejší metodou dochází k postupnému zlepšování úrovně

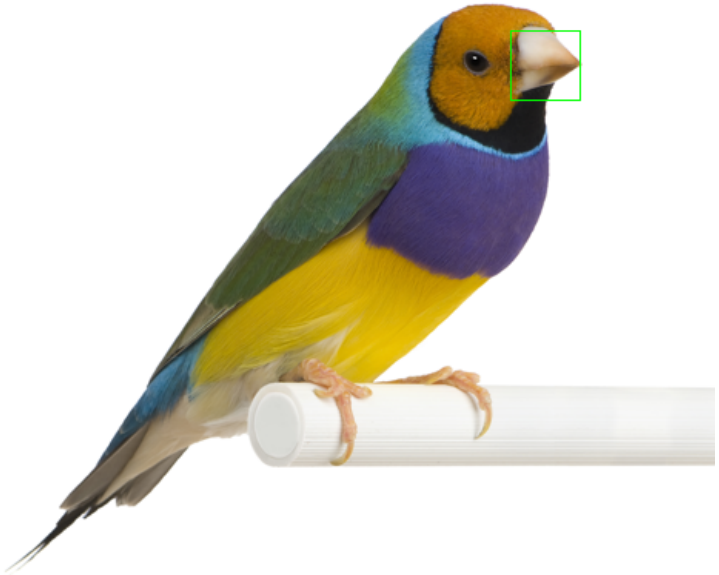
interpolovaného obrazu, zejména na hranici objektu.
Přestože bikubická interpolace produkuje z pohledu lidského vnímání obrazu nejlepší výsledky, byla jako metoda pro předzpracování (přesněji zmenšení) obrazu zvolena bilineární interpolace. Jeví se totiž jako vhodný kompromis mezi rychlostí a výkonem. Navíc u zmenšování obrazu nejsou nedostatky tolik patrné (Obr.5 záměrně zveličuje tyto nedostatky přiblížením části zmenšeného snímku).

3.2 Segmentace

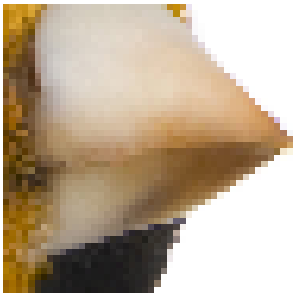
Segmentace obrazu[5] obnáší rozdělení obrazu na oblasti, které silně korelují s reálnými objekty či oblastmi obsaženými v obraze. Nejčastěji se jako rozhodující atribut používá hodnota intenzity pro jednobarevné obrazy a jednotlivé komponenty barev (např. každý z kanálů RGB barevného schématu) pro vícebarevné obrazy[8].
Neexistuje žádná ucelená teorie o segmentaci obrazu[8], následkem čehož nevznikla pouze jediná univerzální metoda pro tento problém. Existuje větší množství metod, které vznikly jako řešení určitého problému a postupně získaly na popularitě. Protože existuje velké množství segmentačních metod, je třeba nějak hodnotit jejich výsledky. Haralick a Shapiro[10] stanovili, že segmenty by měly splňovat následující vlastnosti

- Oblasti segmentů by měly být uniformní a homogenní
- Vnitřky segmentů by neměly obsahovat mnoho malých děr
- Hranice každého segmentu by měly být jednoduché, prostorově přesné a pokud možno nepříliš členité
- Sousední segmenty by měly být co nejvíce rozdílné (s ohledem na segmentační atribut, podle kterého je oblast segmentu uniformní)

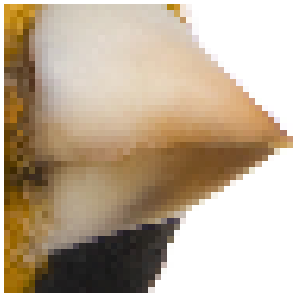
Jeden ze způsobů jak hodnotit segmentační metody předpokládá, že správná segmentace je známa předem. Výsledky měřené metody jsou potom porovnávány s touto "ground truth". Tento postup je však velmi pracný[5] a navíc pro vzevrubné posouzení metody, je třeba mít objemnou databázi takovýchto obrazů (nejznámější je nejspíše The Berkeley Segmentation Dataset [11]).
Dalším ze způsobů je hodnocení bez dohledu (unsupervised evaluation), který je ovšem zpravidla testován na syntetických datasetech a na vlastnosti obrazů zavádí restrikce, které často nemohou být použité v aplikacích z reálného světa.[5]



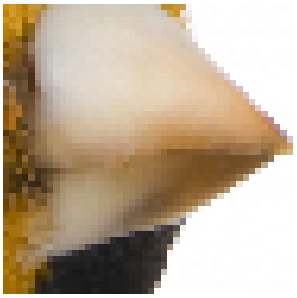
(a) Původní obraz



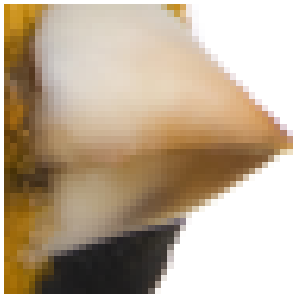
(b) Nejbližší soused



(c) Bilineární int.



(d) Bell



(e) Catmul-Rom

Obr. 8 Porovnání interpolačních metod

Momentálně však neexistuje žádný konsensus o tom, jak tyto metody hodnotit. Pro praktické použití se zpravidla zodpovídají tři otázky[5]

1. Jak často metoda selže (tzn. metoda nedá rozumný výsledek)

2. Jak přesná metoda je

3. Do jaké míry je metoda reprodukovatelná na úspěšných případech

3.2.1 Zvažované metody

Jak již bylo zmíněno výše, existuje nepřeberné množství metod, které dokáží obraz segmentovat. Prvním zvažovaným algoritmem byla, na základě doporučení vedoucího práce, segmentace s využitím hledání minimálního řezu (maximálního tok) grafu - GrabCut[12]. Tato metoda je inicializována nalezením bodů náležících pozadí a popředí, které slouží jako pevné omezení (hard constraint); dodatečná flexibilní omezení (soft constraints) mohou být zavedena, aby reflektovala informaci o oblastech nebo hranicích objektů. Vzhledem k tomu, že tato metoda segmentuje obraz pouze na pozadí a popředí a že požadovaný algoritmus by měl být plně automatický, byla nakonec segmentace s využitím hledání minimálního řezu grafu zamítnuta.
Dalším zvažovaným algoritmem bylo velmi rozšířené shlukování pomocí k-means[5], které lze popsat následovně

1. V obrazu je vybráno (v nejprimitivnější verzi náhodně) k bodů - centroidů

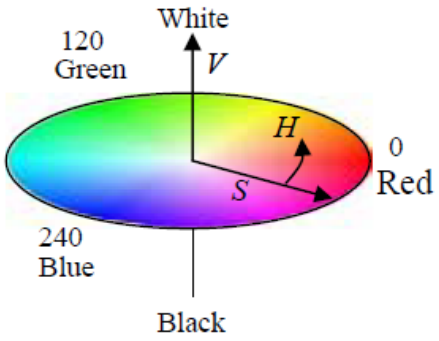
2. Každý bod obrazu je přiřazen ke svému nejbližšímu středu (nejbližšímu ve smyslu zvolené metriky, často používanou je euklidovská vzdálenost)

3. Pro každý takto vzniklý shluk je průměrováním vypočten nový centroid a shlukování začína od 1. kroku

K-means iteruje dokud nedojde ke změně žádného shluku (ideální případ) nebo dokud není překročen počet předem daných iterací. Úskalím této metody je volba parametru k, který udává celkový počet shluků. Existují sice metody na automatické zjištění k (například pomocí detekce hran [13] nebo porovnáním segmentů pro různá k[14]), zavádějí ovšem další vrstvu komplexity do celého algoritmu. Navíc k-means velmi často konvergují k lokálnímu optimu, čímž dochází ke ztrátě počtu segmentů a nepřesnostem v segmentaci.
Nakonec byl zvolen algoritmus zdolávání kopců (Hill-Climbing), který dosáhne segmentace hledáním lokálních maxim v trojrozměrném histogramu[15].

3.2.2 Barevný model HSV

Dobrý barevný model pro segmentaci obrazu by měl splňovat vlastnost, že vnímaná rozdílnost barev odpovídá jejich euklidovské vzdálenosti v tomto modelu. HSV barevný model tuto vlastnost splňuje a navíc velmi dobře odpovídá lidskému vnímání barev[15]. Barevný model HSV je podobně jako RGB třísložkový - H (hue) je odstín barvy, S (satruation) sytost barvy a V (value) představuje jas v porovnání s bílou barvou.



Obr. 9 Barevný model HSV

Na Obr.9[15] je barevný HSV model znázorněný - H jako hodnota na barevném kotouči, S určuje pozici na kotouči od středu a V je pozice barevného kotouče na ose černá-bílá.

RGB hodnoty se na HSV převedou podle následných vztahů[16]

$$R' = \frac{R}{255}, \quad G' = \frac{G}{255}, \quad B' = \frac{B}{255}$$

$$C_{max} = \max\{R', G', B'\}$$

$$C_{min} = \min\{R', G', B'\}$$

$$\triangle = C_{max} - C_{min}$$

$$H = \begin{cases} 0 & \triangle = 0 \\ 60\left(\dfrac{G' - B'}{\triangle} \bmod 6\right) & C_{max} = R' \\ 60\left(\dfrac{B' - R'}{\triangle} + 2\right) & C_{max} = G' \\ 60\left(\dfrac{R' - G'}{\triangle} + 4\right) & C_{max} = B' \end{cases}$$

$$S = \begin{cases} 0 & C_{max} = 0 \\ \dfrac{\triangle}{C_{max}} & C_{max} \neq 0 \end{cases}$$

$$V = C_{max}$$

3.2.3 Algoritmus zdolávání kopců (Hill-Climbing)

Vstupem zvoleného segmentačního algortimu[15] je pole s RGB hodnotami pixelů zpracovávaného obrazu, kde pořadí v poli odpovídá umístění pixelu v obraze (viz. Obr.10). Výstupem je pole *clusters* stejné velikosti jako vstup, které obsahuje rozřazení pixelů do segmentů - $clusters(i) = j$ znamená, že i-tý pixel náleží j-tému shluku.

0	1	2	3
4	5	6	7
8	9	10	11

Obr. 10 Znázornění indexace pixelů (obrázek o rozměrech 4x3)

RGB hodnoty jsou převedeny do HSV a upraveny tak, aby nejmenší hodnota každé složky byla 0 a největší hodnota každé složky 1 (tedy nafitování do intervalu $\langle 0, 1 \rangle$) a poté je pomocí Hill-Climbing algoritmu provedená segmentace. Jednorozměrný Hill-Climbing algoritmus pro H složku vypadá následovně.

1. Vytvoření jednorozměrného barevného histogramu.

2. Zdolávání kopce - začíná se na libovolném nenulovém binu (tj. datovém intervalu) a podle následujících pravidel, se hledá vrchol (kopec), tj. lokální maximum v histogramu

   (a) Porovnání počtu pixelů aktuálního binu s jeho levým a pravým sousedem. Je důležité si uvědomit, že H složka je hodnota na barevném kotouči (Obr.6) a tedy levý krajní bin sousedí s pravým krajním binem.

   (b) Pokud mají sousední biny rozdílný počet pixelů, dojde k přesunu vzhůru k binu s větším počtem pixelů.

   (c) Pokud mají sousední biny stejný počet pixelů, dojde k posouvání na další sousedy, dokud nejsou nalezeny biny s rozdílným počtem pixelů. Přesun vzhůru je proveden na bin s větším počtem pixelů.

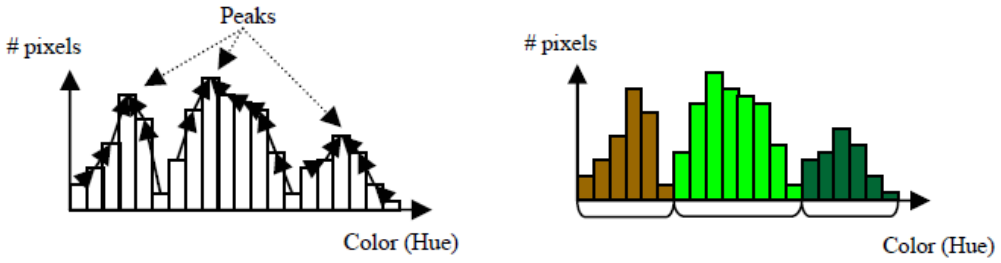
   (d) Postup 2(a)-2(c) je opakován, dokud nedojde k nalezení binu, z kterého již žádným způsobem není možný pohyb vzhůru, tj. sousední biny obsahují menší počet pixelů. Tento bin je indentifikován jako peak (vrchol či kopec), jedná se o lokální maximum v histogramu

3. Je zvolen další libovolný nenulový, avšak dosud nezpracovaný, bin a je zopakován krok číslo 2. Tento krok se opakuje, dokud nejsou zpracovány všechny biny histogramu.

4. Identifikovaná lokální maxima představují počet shluků v obraze (pozn. tato metoda by tedy mohla být jednou z dalších možností automatizace segmentace pomocí k-means shlukování)

5. Jednotlivé biny jsou přiřazeny k tomu lokálnímu maximu, ke kterému se došlo v kroce 2; tímto je segmentace hotova.

Obr.11[15] znázorňuje průběh algoritmu - (a) hledání vrcholů (peaků), (b) přiřazování binů k vrcholům.
Zobecnění tohoto postupu do tří rozměrů (tedy pro všechny tři složky HSV) je přímočaré. V 1. kroce je vytvořen trojrozměrný histogram místo jednorozměrného a ve 2. kroce se pouze liší počet sousedních binů, se kterými se porovnává počet jejich pixelů. Ve třech rozměrech má obecně každý bin místo dvou dvacet šest sousedů (neplatí pro krajní biny histogramu); dále je třeba si uvědomit, že zatímco mezní biny H komponenty spolu sousedí, pro S a V složku toto neplatí. Navíc je třeba zavést následující podmínku - pokud je hodnota S příliš malá (přibližně 0.1), porovnávají se pouze sousední biny ve směru V složky. Když jsou hodnoty S příliš malé, lidské oko nedokáže rozeznat změnu barvy při změně hodnoty V.



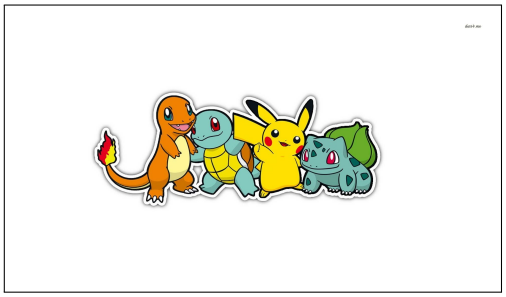
(a) **(b)**

Obr. 11 Hill-Climbing algoritmus

Jedinými parametry tohoto algoritmu je počet jednotlivých binů histogramu. Pravidlem je, že H složka je kvantizována do více úrovní než zbylé dvě složky tak, aby reflektovala různorodost barev. Doporučený poměru binů H:S:V je 16:8:8[15], při tomto rozložení ovšem dochází u větších obrazu k částečnému přesegmentování a zvolen byl proto nakonec poměr 15:7:7.
Pro každý bin histogramu je prozkoumáno všech jeho 26 sousedů a každý pixel obrazu musí být přiřazen k jednomu z lokálních maxim v histogramu (tzn. k segmentu). Při počtu binů N_i a celkovém počtu pixelů N_p je tedy časová složitost Hill-climbing segmentace $O(26N_i + N_p)$.

3.3 První dělení na základě velikosti

Jak je patrné z Obr.12(b) (segmenty znázorněné odstíny šedi), segmentace pomocí Hill-Climbing algoritmu vytváří velmi zrnité oblasti na hranicích jednotlivých objektů v obraze. Jedná se o malé segmenty o velikosti řádově několika desítek až několika stovek pixelů (výjimkou ovšem nejsou ani několikapixelové segmenty). Z hlediska pragmatičnosti nejsou tyto segmenty nijak důležité a je tedy možno je přímo oddělit od dostatečně velikých segmentů. Toto je vykonáno obyčejným prahováním - nejdříve je vypočtena jejich velikost (tj. počet pixelů) a pokud je menší než stanovený práh, segment je "zahozen". Práh byl určen na 0.4% celkového počtu pixelů v obraze, což je při alokaci 250000 pixelů na obraz přibližne 1000 pixelů. Použitím takového prahování došlo u Obr.12 k odstranění čtyřiceti dvou segmentů (odstraněné části jsou znázorněny fialovou barvou na Obr.12(c)).
Při celkovém počtu pixelů N_p a celkovém počtu segmentů k je časová složitost oddělení malých segmentů $O(N_p+k)$ - ke spočítání velikosti stačí jednou projít výstup z Hill-Climbing algoritmu (odtud $O(N_p)$) a poté je každý počet porovnán s prahem (odtud $O(k)$).



(a) Původní obrázek

(b) Segmentace

(c) Odstranění malých segmentů

Obr. 12 První dělení na základě velikosti

3.4 Prostorové oddělení segmentů a druhé dělení na základě velikosti

Segmentace pomocí Algoritmu zdolávání kopců nebere v úvahu prostorové rozložení segmentů, viz jednoduchý příklad na Obr.13 (segmentace na Obr.13(b) opět znázorněna odstíny šedi; v tomto případě pouze černou a bílou barvou, jelikož existují pouze dva segmenty původního obrazu - oranžová a modrá část).

(a) Původní obrázek

(b) Segmentace

Obr. 13 Prostorově neoddělené segmenty

Vstupem algoritmu je pole s rozřazením pixelů do segmentů (po odstranění malých segmentů); výstupem je seznam indexů patřících segmentu (pro každý segment jeden seznam). K prostorovému oddělení je používáno okolí (pro potřeby tohoto textu nazvané *1_4 okolí*) pixelu (bodu) z Obr.14 - o představuje zpracovávaný pixel (bod), x zkoumané okolí.

x	x	x
x	o	

Obr. 14 Okolí pixelu (o) použité při prostorovém dělení

Pro každý segment vypadá pseudokód algoritmu pro prostorové oddělení následovně

0. Inicializace algoritmu:
   $index := 1$
   $rastr :=$ matice o rozměrech původního obrázku, defaultní hodnota 0
   $belong :=$ prázdné pole

1. Pro $\forall$ pixel p zpracovávaného segmentu (p je na pozici *(i,j)* v původním obrázku): *neigh :=* nenulové hodnoty *1_4 okolí* bodu *rastr(i,j)*

   (a) Je-li *neigh* prázdné, potom:
   - $rastr(i,j) := index$
   - *belong.add(index)*
   - $index := index + 1$

   (b) Není-li *neigh* prázdné, potom:
   - $id_{min} :=$ minimální hodnota *neigh*
   - $rastr(i,j) := id_{min}$
   - pro $\forall i \in neigh : belong(i) = id_{min}$

2. Vytvoření prázdného seznamu pro každý prostorově samostatný segment, celkový počet je těchto seznamů je maximální hodnota pole *belong*

3. Pro $\forall l, m$ taková, že $rastr(l,m) \neq 0$ je do z-tého seznamu přidán index odpovídající pozici *(l,m)*; z je hodnota pole *belong* na pozici dané indexem *rastr(l,m)*

Popsaný algoritmus je vysvětlen na následujícím příkladu. Je uvažován obrázek o rozměrech 6x2 a po Hill-Climbing segmentaci zaujímá jeden ze segmentů indexy na pozicích $[2, 4, 6, 7, 8, 10, 11]$, viz. Obr.15 - nalevo jsou křížky vyznačeny body patřící segmentu, napravo je znázorněna indexace jednotlivých pixelů.

		x		x	
x	x	x		x	x

0	1	2	3	4	5
6	7	8	9	10	11

Obr. 15 Příklad oddělovaného segmentu

Inicializace algoritmu (krok 0) pouze vytvoří prázdnou nulovou matici *rastr* o rozměrech 2 řádky a 6 sloupců, vytvoří prázdné pole *belong* a do proměnné *index* přiřadí hodnotu 1.
První pixel segmentu je na pozici 2, *1_4 okolí* tohoto bodu v *rastr* neobsahuje žádnou nenulovou hodnotu, a tak se pokračuje podle kroku 1.(a) - do $rastr(0, 2)$ je přiřazena hodnota 1, do pole *belong* je přidána jednička (prozatím jednoprvkové pole) a hodnota proměnné *index* je navýšena na 2.
Další zpracovávaný bod je na pozici 4 a opět jsou v jeho *1_4 okolí* samé nuly. Na $rastr(0, 4)$ je přiřazena hodnota 2, *belong* je dvojprvkové pole s s hodnotami 1 a 2 a *index* je zvětšen na 3.
Příliš se toho nezmění ani při zpracování dalšího bodu (pozice 6) - *belong* je nyní tříprvkové pole $[1, 2, 3]$ a *rastr* je na Obr.16

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

Obr. 16 Hodnoty matice *rastr* po třech krocích algoritmu

Prvním "zajímavým"pixelem je bod na pozici 7, v jeho *1_4 okolí* jsou 1 a 3 (mimo dvou nul, které jsou ovšem ignorovány). Postupuje se podle kroku 1.(b), do id_{min} je přiřazena menší z hodnot, tedy 1; na *rastr(1,1)* je přiřazena tatáž hodnota a dojde i k úpravě pole *belong*, jež nyní vypadá následovně: $[1, 2, 1]$ (Pozn.: je možné si všimnout malé nesrovnalosti v indexacích, zatímco matice *rastr* a všechna prozatím zmiňovaná pole začínají indexem 0, pole *belong* začíná indexem 1. Tato indexace, ač možná matoucí, je záměrná.)
Hodnoty *rastr* po dokončení 1. kroku algoritmu jsou ve Obr.17; pole *belong* se již nezměnilo - $[1, 2, 1]$. Význam tohoto pole spočívá v přiřazení různých hodnot matice *rastr* prostorově souvislému shluku. Hodnota 1 na třetím indexu pole *belong* znamená, že indexy v *rastr* s hodnotou 3 patří do stejného shluku jako indexy s hodnotou 1; na druhou stranu indexy s hodnotou 2 tvoří samostatný shluk.

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

Obr. 17 Konečné hodnoty matice *rastr*

Následně jsou podle 2. kroku vytvořeny dva prázdné seznamy a podle 3. kroku jsou do nich přidělovány indexy. Výsledkem jsou tedy segmenty popsané indexy $[2, 6, 7, 8]$ a $[4, 10, 11]$.
Nežádoucím vedlejším produktem prostorového oddělování je možnost tvorby příliš malých segmentů. To se může stát v případě, že segment projde prvním testem na minimální velikost a následně je v tomto kroku rozdělen na dvě či více částí, které by tím samým testem již neprošly. Proto je třeba znovu otestovat velikosti a malé segmenty odstranit - opět je nastavena hranice 0.4% celkového počtu pixelů (při alokaci 250000 pixelů na obraz přibližně 1000 pixelů). Zároveň jsou odstraněny (přesněji vzato jsou uloženy "bokem", protože je těchto segmentů potřeba v určitých situacích, viz. dále) příliš velké segmenty, které mohou být považovány za patřící pozadí. V prvním dělení podle velikosti nemohlo k tomuto úkonu dojít, neboť nebylo možné rozeznat příliš velké celistvé segmenty od příliš velkých necelistvých segmentů, tj. takových, které mají po prostorovém oddělení přijatelnou velikost. Práh maximální velikosti byl určen na 15% celkového počtu pixelů v obraze, což dělá přibližně 37500 pixelů při nastavené alokaci.
Algoritmus pro každý segment vytvoří jednu matici o velikosti původního obrázku - prvky matice jsou v 1. kroku sekvenčně procházeny, je kontrolováno jejich *1_4 okolí* a upravována jejich hodnota. Zjištění *1_4 okolí* není závislé na celkovém počtu pixelů ani na počtu segmentů a lze tedy získat v konstantním čase. K úpravě pole *belong* dochází (i když zpravidla tomu tak není) v každé iteraci 1. kroku algoritmu, tj. pro každý prvek matice - jsou měněny maximálně čtyři jeho hodnoty (tolik je maximum nenulových hodnot v *1_4 okolí* každého bodu). Při velikosti tohoto pole N_m, celkovém počtu pixelů N_p a počtu zpracovávaných segmentů k je tedy časová složitost 1. kroku algoritmu $O(4kN_mN_p)$.
2. krok algoritmu je očividně časově konstantní, dochází v něm pouze ke stanovení počtu oddělených segmentů. Ve 3. kroku algoritmu dochází k opětovnému sekvenčnímu procházení (a opět pro každý zpracovaný segment) a k přiřazování indexů jednotlivým segmentům. Časová složitost 3. kroku je tedy $O(kN_p)$.
Celková časová složitost prostorového oddělení je $O(kN_p(4N_m + 1))$, kde $k \ll N_p$ a $N_m \ll N_p$. k je zpravidla v řádech jednotek až několika málo desítek; N_m je zpravidla také v řádech jednotek až několika málo desítek a je závislé na členitosti hranic segmentů.
Složitost odstranění segmentů nežádoucích velikostí je rovna $O(k)$ - každý segment má svůj vlastní seznam, stačí tedy pouze zkontrolovat velikost každého seznamu a porovnat s nastavenými prahy.

3.5 Kritérium pro vybírání segmentů

Cílem algoritmu popsaného na začátku této kapitoly je vybrat z libovolného vstupního obrázku části tak, aby tyto části odpovídaly objektům v obraze. Toho se dosáhne výše popsaným segmentačním algoritmem. Vybrané části by dále měly být vizuálně přijatelné, a i když se jedná o vcelku vágní a hlavně dosti subjektivně založený požadavek, existují deskriptory geometrických tvarů (shape descriptors), podle kterých je možné rozhodnout.
Zvažovány byly následné deskriptory založené na oblasti popisovaného geometrického tvaru (region-based shape descriptors[5])

- Eccentricity — udává poměr délky hlavní osy (major axis) a vedlejší osy (minor axis)
- Elongatedness — udává podobnost tvaru přímce a spočítá se jako poměr šířky a výšky nejmenšího obklopujícího obdélníku (minimum area enclosing rectangle) daného tvaru
- Rectangularity — měří podobnost obdélníku a je vypočtena jako poměr obsahu daného tvaru ku součinu rozměrů nejmenšího obklopujícího obdélníku
- Compactness — vyjadřuje podobnost geometrického tvaru a kružnice

Elongatedness nelze snadno (pomocí nejmenšího obklopujícího obdélníku) vypočítat pro více zakřivené tvary, což z ní dělá nevhodného kandidáta pro popis „dobrých“ tvarů. Podobnost obdélníku (rectangularity) či "zploštění"tvaru (eccentricity) byly také zamítnuty jako vlastnosti, které nepopisují vhodnost tvaru. Vybrána byla tedy kompaktnost; podobnost kružnici (kruhu) se totiž jevila jako vizuálně uspokojivá.

3.5.1 Výpočet kompaktnosti geometrického tvaru

Kompaktnost geometrického tvaru (také někdy nazývána shape index) je numerická kvantita vyjadřující kompaktnost tvaru (a nebo jako bylo zmíněno dříve, podobnost tvaru a kružnice). Kompaktnost je uznávána jako jedna z nejzajímavějších a nejdůležitějších vlastností geometrického tvaru a je hojně používána nejen v odvětví počítačového vidění[17]. Mezi příklady použití patří definování a analýza homogenních oblastí výskytu v ekologii, object matching

a rozpoznávání vzorů (pattern recognition) v oblasti umělé inteligence, popis a vyhledávání objektů v obrazových databázích[17]. V psychologických studiích byla kompaktnost zavedena jako ukazatel stability a estetičnosti geometrického tvaru[18], což jen umocňuje její výběr pro selekci "dobrých"tvarů. Snahy o vyjádření kompaktnosti geometrického tvaru mají dlouho historii([17]). V roce 1822 navrhl Ritter vyjádření kompaktonosti jako poměr obvodu P ku ploše tvaru A. I když je takové vyjádření přímočaré, tento jednoduchý poměr se změní při změně velikosti tvaru. Tuto veličinu je možné učinit bezrozměrnou, pokud se vypočte poměr plochy ku druhé mocnině obvodu; mezi mnohé varianty tohoto postupu patří například poměr $4A/P^2$ (Miller, 1953) či $2\sqrt{\pi A}/P$ (Richardson, 1961). Nejpoužívanějším vyjádřením kompaktnosti tohoto typu je potom *IPQ* index (Osserman, 1978) daný předpisem

$$C_{IPQ} = \frac{4\pi A}{P^2}$$

Hodnoty C_{IPQ} náleží intervalu $(0, 1\rangle$. Geometrické tvary s vyšší hodnotou C_{IPQ} jsou kompaktnější než tvary s nižší hodnotou C_{IPQ}, nejkompkatnější je kruh s C_{IPQ} rovným jedné.
Dalším způsobem číselného vyjádření kompaktnosti je porovnávání s referenčními tvary. Cole v roce 1964 navrhl porovnávání plochy A zkoumaného tvaru s plochou nejmenší opsané kružnice tomuto tvaru A_{SC} jako alternativu ke Gibbsově (1961) poměru $4A/L^2$, kde L je vzdálenost dvou nejvzdálenějších bodů na obvodu tvaru. V roce 1984 zavedli Kim a Anderson toto porovnání jako index DCM (digital compactness measure)

$$C_{DCM} = \frac{A}{A_{SC}}$$

Stejně jako u C_{IPQ} náleží hodnoty C_{DCM} intervalu $(0, 1\rangle$, kdy nejkompaktnější kruh nabývá opět hodnoty jedna. C_{DCM} není bohužel možné použít na nevyplněné geometrické tvary (tj. tvary s otvory) a navíc není invariantní vůči změně velikosti.
Bottema v roce 2000 navrhl další veličinu využívající referenčních tvarů

$$C_{Bottema} = 1 - \frac{|A \cap A_0|}{A_0}$$

která využívá kruhu stejné plochy jako měřený geometrický objekt, konkrétně velikosti průniku tohoto kruhu a měřeného geometrického objektu (A je povrch

objektu, A_0 povrch kruhu). Další index podobný $C_{Bottema}$ představil v roce 2000 Wentz. Jeho podoba je (při stejné notaci jako u $C_{Bottema}$)

$$El = \frac{|A \cap A_0|}{|A \cup A_0|}$$

$C_{Bottema}$ i *El* lze použít na objekty s otvory; nevýhodou je však potřeba nalezení optimálního (tj. maximálního) překrytí měřeného objektu a zkonstruovaného kruhu a navíc opět není ani jeden index invariantní vůči změně velikosti.
Bribiesca v roce 1997 navrhl index NDC (normalized discrete compactness) přímo pro rastrová data

$$C_{NDC} = \frac{\frac{4n-p}{2} - n + 1}{n - 2\sqrt{n} + 1}$$

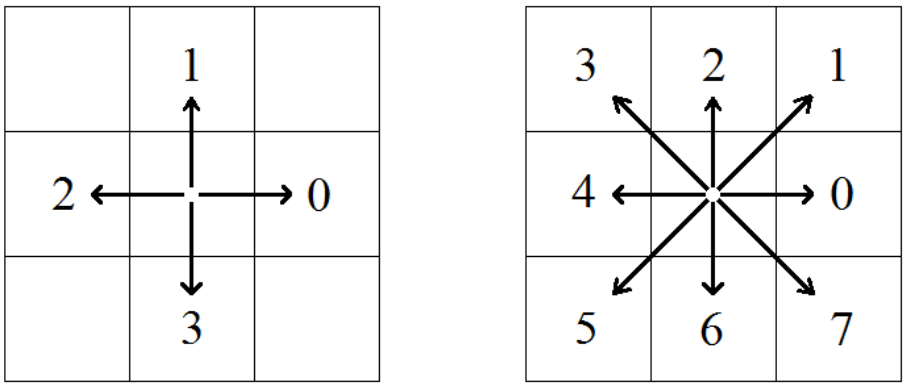
kde p je počet (mezních) hran měřeného geometrického tvaru a n celkový počet pixelů tohoto tvaru. C_{NDC} je invariantní vůči změně velikosti a reflektuje nevyplněnost objektů, je však potřeba určit počet hran p měřeného objektu.
Z výše popsaných veličin na měření kompaktnosti geometrického objektu byl vybrán index C_{IPQ}. Nejen, že lze relativně snadno vypočítat, nemá žádné zásadní nevýhody (dokáže si poradit s dírovanými objekty a je invariantní vůči změně velikosti).

3.5.2 Výpočet IPQ indexu

Pro připomenutí, předpis pro výpočet C_{IPQ} indexu dvojrozměrného geometrického obrazce při známém obvodu P (z anglického perimeter) a při známém obsahu A (area) je

$$C_{IPQ} = \frac{4\pi A}{P^2}$$

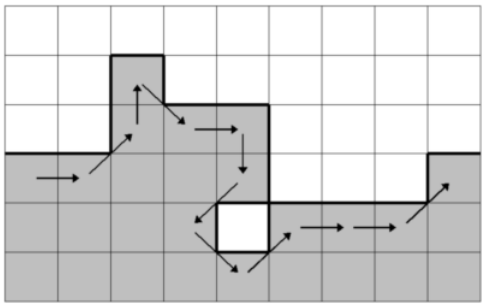
Hodnota A se pro jednotlivé segmenty snadno získá jako počet prvků v seznamu indexů reprezentujícího daný segment, který byl získán při prostorovém oddělování (poslední prováděný úkol); trochu problematičtější je to s určením obvodu P.
K výpočtu obvodu geometrického obrazce je nejprve potřeba zjistit hranici objektu, k čemuž byl použit Freemanův řetězový kód (Freeman's chain code nebo jen Freeman's code)[19]. Tento kód slouží k popisu hranic objektů a využívá k tomu označení okolních pixelů zkoumaného pixelu čísly viz Obr.18. Pro účely této práce byla použita 8-kontektivita (z Freemanova řetězového kódu také vznikl název *1_4 okolí* používaný v kapitole o oddělování segmentů).



(a) 4-konektivita **(b)** 8-konektivita

Obr. 18 Freemanův řetězový kód

Od počátečního pixelu, který je prvním a zároveň i posledním prvkem řetězového kódu, je sekvencí (řetězem) takovýchto čísel popsána hranice objektu zpravidla ve směru hodinových ručiček. Například řetězový kód $[0, 1, 2, 7, 0, 6, 5, 7, 1, 0, 0, 1]$ popisuje část hranice na Obr.19[20].



Obr. 19 Příklad popisu hranice objektu

Algoritmus pro nalezení kódu[21] začíná v jednom z extrémních pixelů, tzn. v jednom z pixelů ležících nejvíce vlevo, nejvíce vpravo, nejvíce dole nebo nejvíce nahoře. Jelikož první pixel v seznamu každého segmentu je z podstaty algoritmu na prostorové oddělení zároveň pixelem ležícím nejvíce nahoře, je zvolen právě tento pixel. Další pixely segmentu mohou sice ležet ve stejné výšce (a to pouze napravo od prvního pixelu seznamu), pro zjištění řetězového kódu to ovšem není žádnou překážkou. Ze zvoleného počátečního bodu může řetěz pokračovat pouze ve směrech 0, 7, 6 a 5; směry 1, 2 a 3 jsou vyloučeny, jelikož žádný pixel nemůže ležet nad startovní pozicí a směr 4 je vyloučen na

základě argumentu z předchozí věty, tj. startovní pozice nemůže z principu mít souseda vlevo. Je třeba dodržovat i pořadí prohledávaných sousedních směrů - ze startovního pixelu je třeba nejprve prozkoumat pixel ve směru 0 a teprve pokud tento pixel nenáleží segmentu, pokračuje se ve směru 7 (a následně 6 a 5, je-li to nutné). Po nalezení správného pixelu z hranice segmentu se stejným způsobem pokračuje v získávání dalších částí kódu, dokud se algoritmus nevrátí opět na začátek. V každém kroku jsou ovšem prozkoumávány jiné směry (tzn. 0, 7, 6, 5 nejsou univerzální posloupností kandidátů); první takovýto směr je o dva více než poslední přidaný, například je-li posledním prvkem řetězového kódu 4, prvním kandidátem je směr 6 a v případě jeho nevybrání se pokračuje dále ve směru hodinových ručiček (tj. 5, 4, 3 atd.).
Popsaný algoritmus je vysvětlen na následujícím příkladu. Je uvažován obrázek o rozměrech 4x4 a segment popsaný indexy $[1, 2, 4, 5, 6, 8, 9, 10, 11, 13, 14]$, viz. Obr.20 - nalevo jsou křížky vyznačeny body patřící segmentu, napravo je znázorněna indexace jednotlivých pixelů.

	x	x	
x	x	x	
x	x	x	x
	x	x	x

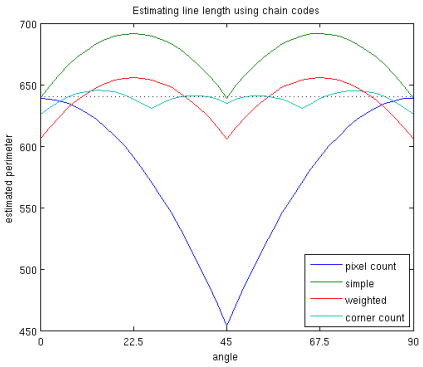
0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

Obr. 20 Příklad segmentu

Začíná se bodem, který má index 1, a postupně jsou prohledávány směry 0, 7, 6 a 5. Přesněji řečeno by tyto směry byly prozkoumávány, již směr 0 ovšem náleží segmentu, je tedy přidán do řetězového kódu a algoritmus pokračuje. Prvním kandidátem na další postup je směr o dva větší než poslední přidaný, tj. 2. Pixel v tomto směru ovšem nenáleží segmentu (neleží ani v obrázku samotném) a tak je prozkoumána následující možnost po směru hodinových ručiček (tj. 1), která je však také zamítnuta. Postupně jsou zavrhnuty i směry 0 a 7. Bod ve směru 6 je již součástí segmentu, a tak je přidán do řetězového kódu, který má nyní tvar 0-6. Algoritmus pokračuje podle stejného principu dále (další krok zvažuje nejdříve směr 0), dokud se obrys objektu neuzavře. Výsledný řetězový kód má podobu 0-6-7-6-4-4-3-2-1.
Po zjištění obrysu (reprezentován řetězovým kódem) je již možno vypočítat (přesněji vzato odhadnout) hodnotu obvodu P, viz. [20]. První možností je jednoduché počítání pixelů, a i když se jedná o velmi přímočarou možnost, dochází k podhodnocení výsledného obvodu; diagonální kroky (lichá čísla v řetězovém kódu) mají totiž ve skutečnosti větší délku než uvažovaných 1. Freeman navrhl pro každý takový diagonální krok přičítat $\sqrt{2}$ namísto 1, viz. [19], což reflektuje skutečnou vzdálenost středů pixelů. Problém reálných objektů a jejich obrysů

spočívá v digitalizaci dat. Je-li součástí hranice rovná přímka (ve smyslu kolmosti k hranicím obrázku), vše funguje tak jak má. Je-li ovšem ta samá přímka mírně natočena (např. o několik málo stupňů), dojde převedením na obraz složený z pixelů k jejímu "zazubatění"a součet vzdáleností středů pixelů tedy neodpovídá její skutečné délce. Proffitt a Rosen[22] toto reflektovali a odhad upravili na $P = 0.948e + 1.340o$, kde e je počet sudých čísel v řetězovém kódu a o je počet lichých čísel v řetězovém kódu. Vossepoel a Smeulders[23] výpočet dále zpřesnili na $P = 0.948e + 1.340o - 0.091c$, kde e a o mají stejný význam jako v předchozím vzorci a c udává kolikrát řetězový kód změnil hodnotu (corner count).

Luengo provedl experiment[20], kdy postupně natáčel obdélník a pro každé natočení měřil každou z metod jeho obvod. Výsledky jsou znázorněny na Obr.55 - na ose y je hodnota odhadnutého obvodu, na ose x natočení obdélníku, modře jsou hodnoty pro počítání pixelů řetězového kódu, zeleně Freemanovo zlepšení, červeně metoda od Proffitta a Rosena, tyrkysově metoda od Vossepoela a Smeulderse a čárkovaně je na ose y skutečná hodnota obvodu. Je vidět, že počítání pixelů podhodnocuje obvod u jakéhokoli natočení a také, že metoda od Vossepoela a Smeulderse se s natočením vypořádá opravdu nejlépe z představených metod.



Obr. 21 Porovnání metod na měření obvodu

Vypočtení IPQ indexu jednoho segmentu je při počtu pixelů v tomto segmentu N_s roven $O(N_s)$. Obsah segmentu A lze získat konstantě, je roven velikosti seznamu reprezentujícího segment. Obvod segmentu P je získán v čase $O(N_s)$ - při počítání řetězového kódu jsou prozkoumány pixely hranice (maximálně N_s) a u každého je v konstantním čase určen směr postupu (8 možných směrů nezávislých na počtu pixelů).

3.6 Operace ovlivňující IPQ index

Jak je patrné z Obr.12, segmenty jsou u svých hranic velmi členité a obsahují poměrně velké množství různě velikých otvorů. Před samotným výběrem žádoucích segmentů s dostatečným IPQ indexem jsou proto tyto vizuální nedostatky odstraněny pomocí dvou operací, které IPQ index mění (zvětšují). Tyto operace jsou vysvětleny v následujících dvou podkapitolách.

3.6.1 Matematická morfologie

První z použitých metod na vizuální zlepšení segmentu se opírá o binární matematickou morfologii[5] (tj. morfologii binárních obrazů), matematický nástroj používající nelineární operátory operující na tvaru objektu. Jelikož se jedná o poměrně složitou problematiku, bude zde diskutována hlavně s důrazem na praktické použití.

Morfologická analýza využívá možnosti zápisu binárních obrazů jako podmnožin dvojrozměrného prostoru celých čísel $\mathcal{Z}^2$. Například segment (což je vlastně binární obraz - pixely segmentu patří obrazu, tj. mají hodnotu 1 a zbývající pixely obrazu nepatří, jejich hodnota je 0) na Obr.22,

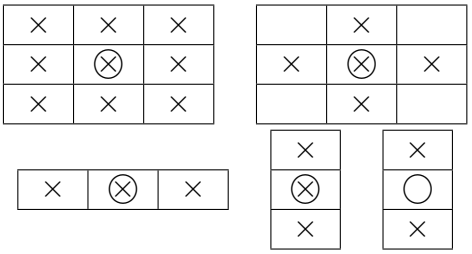
	×	×	
×	×		
◯	×	×	

Obr. 22 Příklad segmentu jako binárního obrazu

kde × představují pixely patřící segmentu a ◯ je počátek souřadnicové soustavy, tj. má souřadnice (0,0), lze vyjádřit jako množinu

$$X = \{(1,0),(2,0),(0,1),(1,1),(1,2),(2,2)\}$$

Morfologická transformace je vyjádřena relací množinově zapsaného binárního obrazu a strukturního elementu - malé množiny vztažené k lokálnímu počátku, která slouží jako "lokální sonda"v morfologických operacích. Nejčastěji používané strukturní elementy jsou ve Obr.23

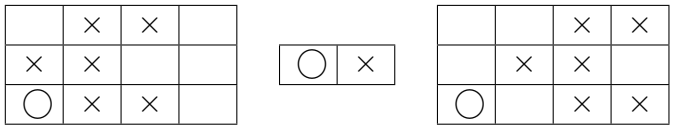


Obr. 23 Nejčastěji používané strukturní elementy

Prvním z pomocných úkonů používaných v binární morfologii je translace X_h množiny X o radiusvektor h daná vztahem.

$$X_h = \{p \in \mathcal{E}^2 : p = x + h \quad \text{pro } \forall x \in X\}$$

kde $\mathcal{E}^2$ označuje 2D euklidovský prostor. Příklad je na Obr.24



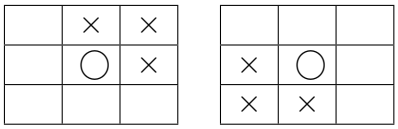
X h X_h

Obr. 24 Posunutí o radiusvektor

Druhou pomocnou operací je transpozice $\breve{B}$ množiny B (někdy označováno jako středová symetrie)

$$\breve{B} = \{-b : \ \forall b \in B\}$$

Příklad transpozice je na Obr.25



B $\breve{B}$

Obr. 25 Transpozice

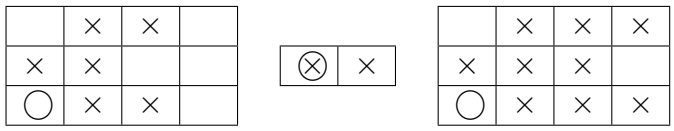
Binární matematická morfologie využívá dvou základních operací, které jsou neinvertovatelné, dilatace a eroze. Binární dilatace $\oplus$ lze vyjádřit jako

$$X \oplus B = \{p \in \mathcal{E}^2 : p = x + b, \ x \in X, \ b \in B\}$$

nebo pomocí Minkowského součtu jako sjednocení posunutých množin

$$X \oplus B = \bigcup_{b \in B} X_b$$

Příklad binární dilatace je na Obr.26



X B $X \oplus B$

Obr. 26 Binární dilatace

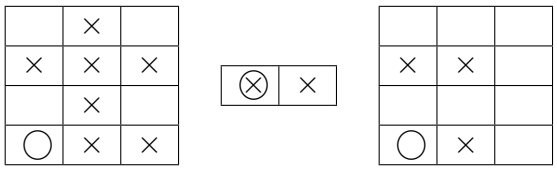
Binární erozi $\ominus$ je také možné zapsat dvěma způsoby. Buď se kontroluje, zda všechna možná posunutí $x + b$ leží v původní množině X a pokud ano, náleží bod x také výsledku (tj. erodované množině)

$$X \ominus B = \{p \in \mathcal{E}^2 : p = x + b \in X \text{ pro } \forall b \in B\}$$

a nebo je eroze určena jako Minkowského rozdíl (průnik všech posunutí množiny X o každý vektor $-b$)

$$X \ominus B = \bigcap_{b \in B} X_{-b}$$

Příklad binární eroze je na Obr.27

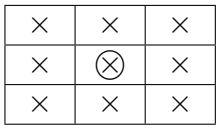


X B $X \ominus B$

Obr. 27 Binární eroze

Na každý segment je použita binární dilatace (se strukturním elementem z Obr.28), čímž dojde k zaplnění drobných děr a úzkých zálivů. Zároveň však dojde k "nakynutí"objektu a z toho důvodu je následně použita binární eroze (s totožným

strukturním elementem). Takovéto posloupnosti operací $X \bullet B = (X \oplus B) \ominus B$ se říká binární uzavření $\bullet$. Výsledkem je segment s menším (nebo v případě „pěkných“ objektů stejným) obvodem a větším (nebo opět v případě „pěkných“ objektů stejným) obsahem než segment původní.

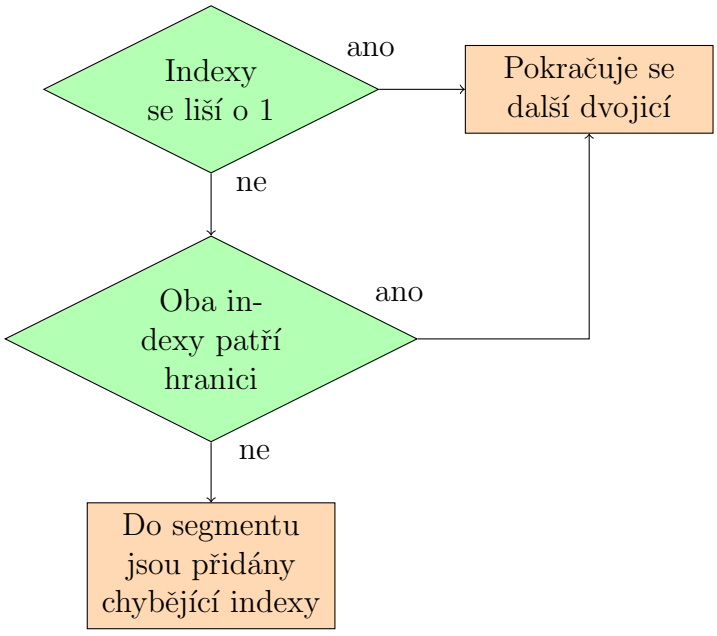


Obr. 28 Použitý strukturní element B

Určení časové složitosti záleží na datové struktuře, zatím vágně označované jako seznam, ve které je segment uchován. V každém případě je třeba provést 8 posunutí segmentu a tato posunutí pro dilataci sjednotit a pro erozi po dvojicích proniknout.

3.6.2 Zaplňování velkých děr

Aplikace matematické morfologie, konkrétně binárního uzavření, se vypořádá mimo členitosti hranic i s malými otvory v objektech, v těch ovšem mohou i nadále zůstat otvory většího charakteru.



Obr. 29 Vývojový diagram vyplňování segmentů

Vývojový diagram algoritmu pro jejich vyplnění je na Obr.29 Sekvenčně je procházen seřazený seznam s indexy segmentu a po dvojicích jsou kontrolovány sousední indexy. Liší-li se tyto indexy pouze o jedna, není zde žádný prostor pro jakýkoli otvor a pokračuje se tedy další dvojicí. Pixely hranice segmentu leží v seznamu na sousedních pozicích a lišit se mohou i o více než jedna, v tomto případě se ovšem také o otvor v objektu nejedná a je tedy opět možno přejít na další dvojici. K vyplnění dojde pouze tehdy, jsou-li sousední indexy rozdílné o více než jeden a alespoň jeden z nich neleží na hranici objektu, tj. leží někde uvnitř. V tomto případě je třeba po jedné doplnit všechny další indexy v rozmezí této dvojice.
Pro algoritmus je tedy nutné zjistit indexy ležící na hranici segmentu. K tomu se využije řetězový kód reprezentující hranici, který je spočítán v těchto místech a dále se použije i pro výpočet IPQ indexu, jelikož vyplnění mezer v objektech nijak nezmění hranici objektu. Jak bylo řečeno dříve, první index v seznamu každého segmentu je také počátečním pixelem pro výpočet řetězového kódu. Toho se využije při zjišťování hraničních indexů, které je možno odvodit z tabulky ve Obr.30 (w je šířka obrázku v pixelech)

hodnota řetězového kódu	0	1	2	3	4	5	6	7
změna indexu	*+1*	*-(w-1)*	*-w*	*-(w+1)*	*-1*	*+w-1*	*+w*	*+w+1*

Obr. 30 Změna indexu v závislosti na hodnotě řetězového kódu

Dále je třeba počítat s nepříliš častou eventualitou, kdy jeden segment může být uvnitř druhého. V tomto případě je vyplnění otvoru patřícímu menšímu segmentu ve větším segmentu nežádoucí. K zjišťování takovéto situace jsou použity pozice mezních pixelů každého objektu, tj. pozice nejvíce vlevo, pozice nejvíce vpravo, pozice nejvíce dole a pozice nejvíce nahoře, podle kterých se dá snadno zjisti zda je segment potenciálně uvnitř jiného či nikoli. Navíc získání těchto mezních pozic nic nestojí, je možno je zjistit jako vedlejší produkt binárního uzavření, při kterém jsou procházeny všechny indexy každého segmentu.
Algoritmus zkoumá každou sousední dvojici v seřazeném seznamu a v případě, kdy se jejich hodnota liší o více než jedna zjišťuje, zda jsou oba pixely hraniční. Seznam s hraničními pixely o velikosti N_b nemusí být seřazen, a tak je třeba ho sekvenčně projít; časová složitost tohoto úkonu je tedy $O(N_b)$. Přidávání chybějících pixelů je při počtu těchto pixelů N_{miss} možné v čase $O(N_{miss})$. Při celkovém počtu pixelů segmentu N_s se tedy časová složitost může vyšplhat na $O(N_{miss}N_bN_s + N_s)$, neboť je třeba ještě vypočítat Freemanův řetězový kód segmentu (složitost diskutována dříve). Navíc většinou platí, že $N_b \ll N_s$

a $N_{miss} \ll N_s$. Zpravidla je ovšem počet vyplňovaných částí v řádu jednotek a algoritmus se tak velmi blíží složitosti $O(N_s)$.
Zjištění, zda jeden segment leží uvnitř jiného, je při celkovém počtu segmentů k možný v čase $O(k(k+1)/2)$ - je potřeba porovnat čtyři mezní hodnoty každé dvojice. Samotné odstranění pixelů menšího segmentu z většího je časově náročné, při velikosti malého segmentu N_{small} a velkého segmentu N_{big} je časová složitost $O(N_{small}N_{big})$ - pro každý pixel menšího objektu je třeba zjistit, zda tento náleží ve větším. Naštěstí k tomuto jevu dochází velmi zřídka.

3.6.3 Výsledky operací

Výsledky obou výše popsaných operací jsou znázorněny na příkladu Obr.31. Na Obr.31(a) je podoba původního segmentu, na první pohled je patrné velké množství různě velikých děr a také členitost jeho hranice viz. Obr.31(b). Aplikace binárního uzavření odstraní velké množství děr a zároveň i vyhladí hranice, viz. Obr.31(c) a Obr.31(d). Stále je však patrná přítomnost otvorů, jež jsou odstraněny algoritmem k tomu určeným Obr.31(e).

3.7 Vybírání segmentů podle IPQ indexu

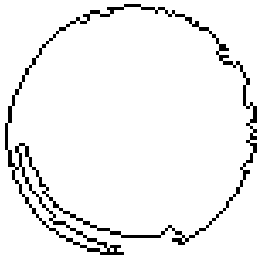
Po úpravě segmentů z předchozí kapitoly je nyní možné vybrat výsledné segmenty. Výběr probíhá ve třech fázích:

1. Prahování s nízkým prahem a zohledněním pozice segmentu v obraze
2. Dělení velkých, tj. příliš vysokých či příliš širokých, segmentů
3. Prahování s vyšším prahem a bez zohlednění pozice segmentu v obraze

3.7.1 Výběr kompaktních segmentů - 1. průchod

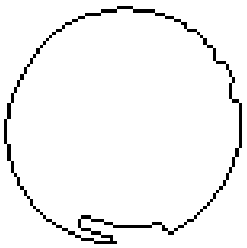
V první fázi je zohledněna pozice segmentu, konkrétně je kontrolováno zda segment neleží v přílišné blízkosti hranic obrázku a pokud ano, je uměle snížen jeho IPQ index. Tento úkon vychází z myšlenky, že „nehezké“ (tj. nepříliš kompaktní) objekty jsou vizuálně přijatelnější, pokud leží uprostřed obrázku. K určení vzdálenosti od hranice je použito těžiště segmentu CoG (center of gravity), které je vypočteno následovně

(a) Původní segment



(b) Hranice původního segmentu

(c) Segment po aplikaci binárního uzavření



(d) Hranice segmentu po aplikaci binárního uzavření

(e) Segment po vyplnění děr

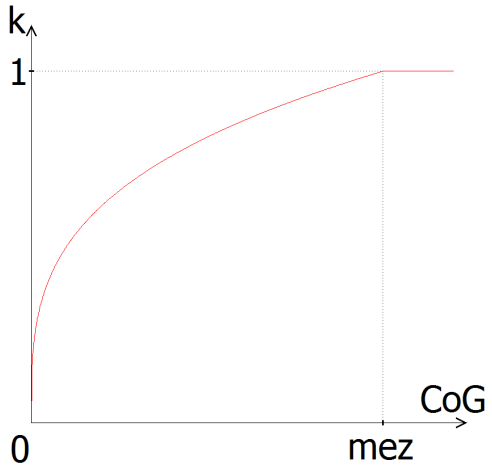
Obr. 31 Zlepšování vlastností segmentu.

$$CoG = (x_0,\ y_0)$$
$$x_0 = \frac{\sum_{i=1}^{A} \left\lfloor \frac{s(i)}{w} \right\rfloor}{A}$$
$$y_0 = \frac{\sum_{i=1}^{A} (s(i) \bmod w)}{A}$$

kde s je seznam s indexy segmentu, A je plocha segmentu (počet prvků s) a w je šířka obrázku.
IPQ index je přenásoben konstantou k, viz. Obr.32

$$k = \begin{cases} d^{0.3} & \text{pro objekty v blízkosti hranic} \\ 1 & \text{jinak} \end{cases}$$

kde d je relativní vzdálenost těžiště objektu od hranice obrázku k hodnotě mez z Obr.32, konkrétně minimum vzdáleností y_0 od levého a pravého okraje a minimum vzdáleností x_0 od horního či dolního okraje. Hodnota mez je 20% celkové šířky, respektive 15% celkové výšky



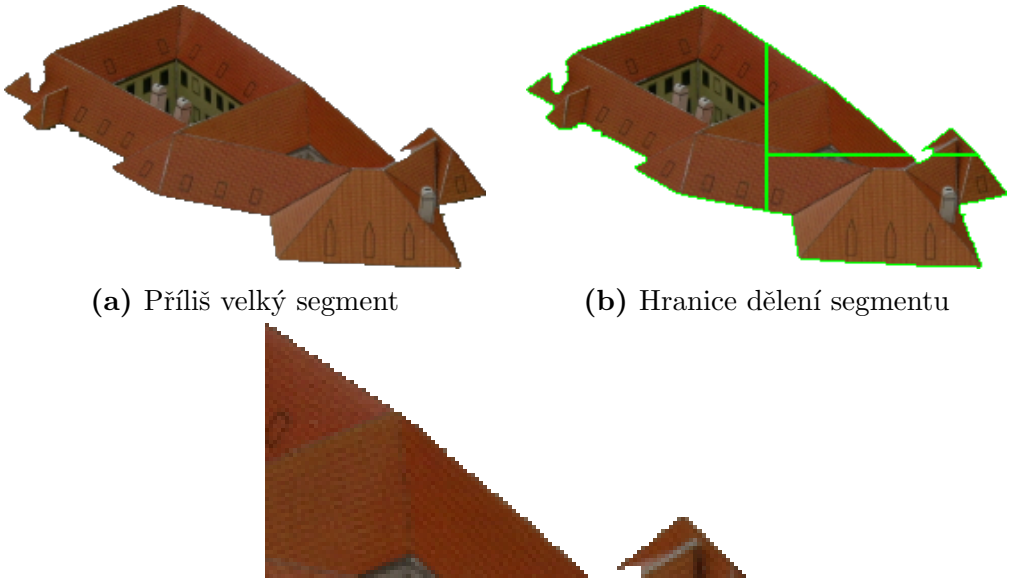
Obr. 32 Úprava IPQ indexu v blízkosti hranic obrázku.

k je počítáno zvlášť pro vzdálenost od levého a pravého okraje a zvlášť pro vzdálenost od horního a spodního okraje. IPQ index je potom přenásoben menší z obou hodnot, jelikož přenásobení oběma by velmi znevýhodňovalo objekty v rozích obrázku.

IPQ index každého segmentu je poté porovnáván s mezní hodnotou 0.22 a segmenty s menším IPQ indexem jsou zahozeny.
Pro každý segment je třeba spočítat jeho těžiště, k čemuž je třeba projít všechny indexy segmentu. Při celkovém počtu pixelů v segmentu N_s je tedy časová složitost této operace rovna $O(N_s)$, jelikož je třeba projít všechny indexy segmentu.

3.7.2 Rozdělení příliš velkých segmentů

Následuje dělení segmentů, které sice splňují kritéria omezující celkovou plochu, avšak jsou buď příliš široké, a nebo příliš vysoké. Mezí pro rozdělování je 35% celkové šířky obrázku a 30% celkové výšky obrázku. Segmenty jsou nejprve rozřezány na šířku a poté podélně. Takovýmto dělením může ovšem dojít k nežádoucímu efektu - tvorbě prostorově nesouvislých segmentů, viz Obr.33. Je tedy třeba opět použít algoritmus představený v kapitole 3.4.



(a) Příliš velký segment **(b)** Hranice dělení segmentu

(c) Prostorově nesouvislý segment

Obr. 33 Nežádoucí efekt dělení velkých segmentů.

Při celkovém počtu pixelů v segmentu N_s je časová složitost dělení $O(2N_s)$ - je třeba nejprve projít indexy při dělení na šířku a přiřadit je nově vytvořenému segmentu a pak je třeba obdobně postupovat při dělení podélném. Celková složitost tohoto kroku je tedy při počtu nových segmentů l rovna $O(2lN_s+lZ)$, kde Z je složitost prostorového oddělení nesouvislých segmentů (diskutována v příslušné kapitole).

3.7.3 Výběr kompaktních segmentů - 2. průchod

Po rozdělení příliš velkých segmentů je aplikováno druhé dělení segmentů podle IPQ indexu. Tentokráte není zohledněna pozice objektu vůči hranicím obrazu, neboť se předpokládá odstranění nežádoucích objektů při prvním prahování. Je však zvýšena hranice pro odmítnutí - nyní již musí IPQ index dosahovat alespoň hodnoty 0.28. Hlavní myšlenkou dvoustupňového prahování je dát šanci velkým objektům, které sice samy o sobě nejsou kdovíjak kompaktní, jejich části však po jejich rozdělení již kompaktní býti mohou.
Časová složitost tohoto kroku je pro jeden segment rovna $O(N_s)$, kde N_s je celkový počet pixelů v segmentu - je totiž třeba přepočítat řetězový kód, který se vlivem případného dělení mohl změnit.

3.8 Tvorba dodatečných segmentů

V případě některých vstupních obrázku je možné, že dosavadním postupem nedojde k vybrání dostatečného množství segmentů, nemusí být vybrán dokonce ani jeden. Například u obrázku tvořeného čtyřmi různobarevnými a stejně velkými čtverci vzniknou sice po segmentaci čtyři kompaktní segmenty, které ovšem neprojdou druhým dělením na základě velikosti (všechny segmenty by měly velikost 25% celkového počtu pixelů, stanovená mez je však 15%).
V takovýchto a podobných situacích je třeba nějak uměle dotvořit další segmenty tak, aby jejich celkový počet byl přijatelný. K tomu je použit postup z Obr.34. Nejprve jsou prozkoumány takové segmenty, které neprošly prahováním jejich IPQ indexu. Konkrétně je snížena mez přijatelnosti z 0.28 na 0.18. Nejedná se tedy prozatím o tvorbu nových výřezů.
Pokud ani snížení prahu IPQ indexu nepřinese kýžený počet segmentů, jsou použity velké segmenty, které byly odstraněny dříve (kapitola 3.4). Nejsou však použity celé, dojde pouze k vyříznutí jejich částí. Velikost těchto výřezů je 12% procent celkové šířky obrazu na 12% celkové výšky obrazu (zpravidla se tedy jedná o obdélníky). Tvorba jednoho takového výřezu probíhá následovně:

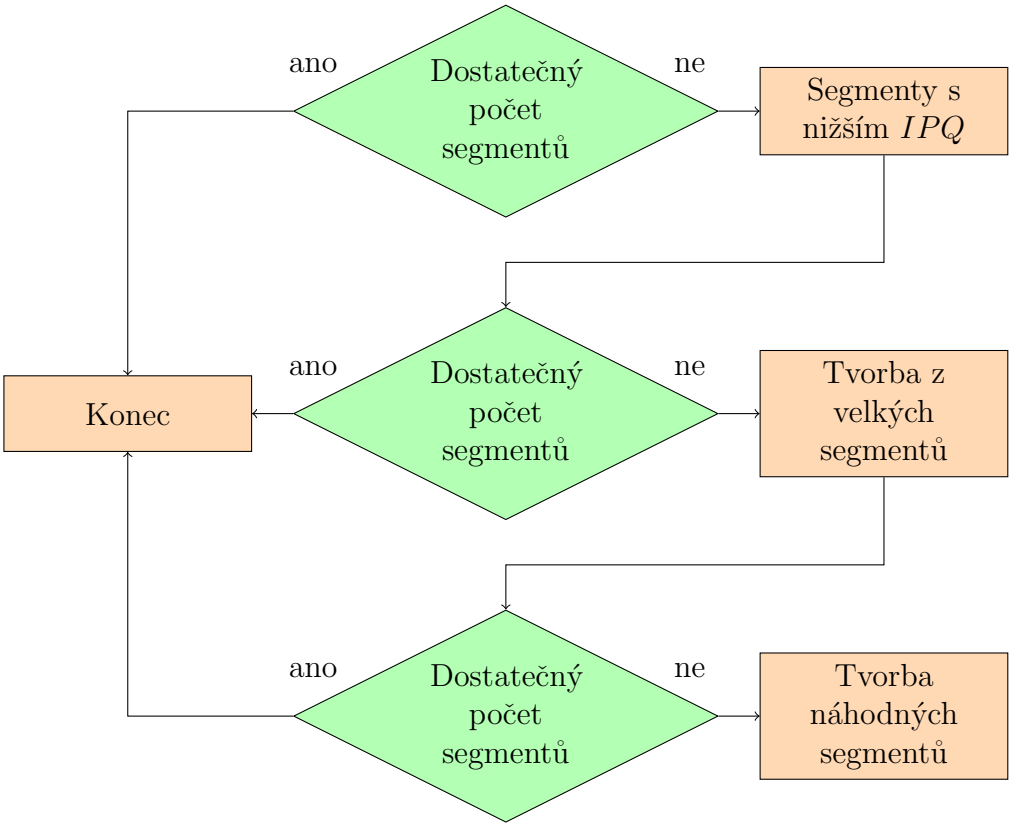
1. Náhodná volba umístění výřezu (samozřejmě tak, aby ležel uvnitř příslušného velkého segmentu)

2. Kontrola přípustnosti výřezu - protíná-li se výřez s již existujícím segmentem nebo leží-li výřez příliš u okraje obrázku (tj. část výřezu by měla být mimo obrázek), pokračuje se 1. krokem.

3. Kontrola velikosti výřezu - je-li výřez příliš malý (opět menší než 4% z celkového počtu pixelů), pokračuje se krokem číslo 1. Tento případ může nastat, jelikož k odstranění velkých segmentů došlo před vyplňováním

děr. Výřez tedy může být velmi děrovaný a navíc může ležet u okraje velkého segmentu, z kterého je vytvářen, což znamená, že může být menší než požadovaných 12% obou rozměrů.

4. Vyplnění výřezu (stejný postup jako v kapitole 3.6.2) a přidání do výsledku.



Obr. 34 Vývojový diagram tvorby dodatečných segmentů

Snahou je dotvořit vyřezáváním z velkých segmentů dostatečný počet segmentů. Z principu výše popsaného postupu se to však nemusí vždy podařit. Například existuje-li jen jeden velký segment, který není nikterak obrovský, a z kterého je třeba vytvořit několik výřezů, záleží uskutečnitelnost tohoto úkonu na jejich umísťování, které je ovšem náhodné. Může se tedy stát, že další výřez již není možné vybrat (v 1. kroku není možné jakkoli vybrat jeho umístění tak, aby se neprotínal s již vytvořenými výřezy) a popsaný algoritmus se zacyklí. Z tohoto důvodu je počet pokusů na tvorbu výřezu z velkého segmentu omezen.
V případě, že ani po vyřezávání z velkých segmentů není dostatečný počet výsledků, je použito vyřezávání z obrázku bez omezení. Princip je téměř totožný

jako u postupu v předchozím odstavci, neexistuje však omezení na umístění výřezu spjaté s příslušností k segmentu a výřezy jsou již vyplněné. Při rozumném minimálním počtu výsledných segmentů (v řádu jednotek, konkrétně byla použita mez rovna pěti) nedojde ani k zacyklení algoritmu a je tedy vždy možné vybrat výřez. Třetí fáze dodatečné tvorby však neprodukuje vizuálně celistvé výřezy.

3.9 Finální úprava segmentů

Na začátku algoritmu byl kvůli časovým úsporám vstupní obraz zmenšen, což znamená, že i výsledné segmenty jsou zmenšené a je tedy třeba je zvětšit na původní velikost. Opět je použita bilineární interpolace a pro každý segment je postupováno následovně:

1. Vytvoření binárního obrazu ze segmentu. Prozatím byl segment reprezentován seznamem a je tedy potřeba z něj vytvořit binární obraz. Například ze segmentu $[1, 2, 5]$ vznikne (při velikosti původního obrázku 4 x 2) binární obraz $[0, 1, 1, 0, 0, 1, 0, 0]$.

2. Bilineární interpolací je tento binární obraz převeden zpět na původní velikost.

3. Indexy ve zvětšeném obraze s nenulovou hodnotou patří zvětšenému segmentu (opačným způsobem než v 1. kroku je z obrazu vytvořen seznam).

Rychlejším postupem by zajisté bylo vytvořit jeden „binární“ obraz ze všech segmentů, kde každému segmentu by bylo přiřazeno rozdílné celé číslo. Problém by ovšem nastal u segmentů, které se před zvětšením dotýkají. Například u dotýkajících se segmentů číslo 1 a 7 se po zvětšení na jejich společných hranicích mohou vyskytovat hodnoty v intervalu $\langle 1, 7\rangle$ a není tedy jasné, jak tyto hraniční pixely přiřazovat.
Doposud byl seznam každého segmentu obyčejným výčtem indexů, které segmentu náležely. Pro zmenšený obrázek se nejednalo o podstatný problém, pro segmenty obrázku původní (větší) velikosti dochází však k tvorbě obrovských výčtů a přitom je většina obsažené informace přebytečná. K popisu segmentů je tedy použita datová struktura, která popisuje souvislé části jako dvojici ve formátu „od-do“. Přesněji se jedná o pole $[Z_1, K_1, Z_2, K_2, Z_3, K_3, ...]$, kde dvojice Z_i, K_i znamená, že segment obsahuje všechny hodnoty v intervalu $\langle Z_i, K_i\rangle$.Tato struktura je dále vysvětlena na následujícím příkladu.
Je uvažován obrázek s rozměry 5x5, viz Obr.35, a segment daný výčtem indexů $[0, 1, 2, 5, 6, 7, 11, 12, 14, 15, 16, 17, 18, 19, 21, 22, 23, 24]$. Tento segment lze

×	×	×		
×	×	×		
	×	×		×
×	×	×	×	×
	×	×	×	×

Obr. 35 Příklad segmentu jako binárního obrazu

úsporněji zapsat jako $[0, 2, 5, 7, 11, 12, 14, 14, 15, 19, 21, 24]$, kde například dvojice 15,19 znamená: segment obsahuje všechny indexy počínaje 15, konče 19.

3.10 Rozmazání vyseparovaných částí ve vstupním obrázku

Poslední fází algoritmu je označení vyseparovaných oblastí ve vstupním obrazu, k čemuž je použito rozmazání obrazu. Přesněji je vytvořen nový obraz, který je rozmazanou variantou původního a části v původním obraze patřící segmentu jsou nahrazeny odpovídajícími částmi z této rozmazané varianty.
K rozmazání obrazu používá konvoluce $*$, kterou lze vyjádřit vztahem[5]

$$(f * h)(i, j) = \sum_{m,n \in \mathcal{O}} h(i - m, j - n) f(m, n)$$

kde $\mathcal{O}$ je okolí současné pozice, h je konvoluční jádro (též konvoluční maska). Při volbě Gaussovy funkce jako jádra (také označováno jako Weierstrassova transformace)[24] dojde k požadovanému rozmazání. Při použití takového jádra o poloměru r je časová složitost této operace při celkovém počtu pixelů n rovna $O(r^2 n)$[25].
K urychlení této operace lze využít separability výše zmíněného dvojrozměrného jádra, tzn. že jádro lze vyjádřit jako součin dvou jednorozměrných jader[26] a poté postupně použít jednorozměrnou konvoluci[27](tj. nejdříve je použita jednorozměrná konvoluce podle řádků a na výsledek je použita jednorozměrná konvoluce podle sloupců) danou vztahem

$$\begin{aligned}(f * h)(i) &= \sum_{m \in \mathcal{O}} h(i - m) f(m) \\ &= \sum_{m \in \mathcal{O}} h(i) f(i - m)\end{aligned}$$

Časová složitost se zmenší na $O(2rn)$[24]. K dalšímu zrychlení lze využít faktu, že aproximovat konvoluci gaussovským filtrem lze opětovným konvolováním

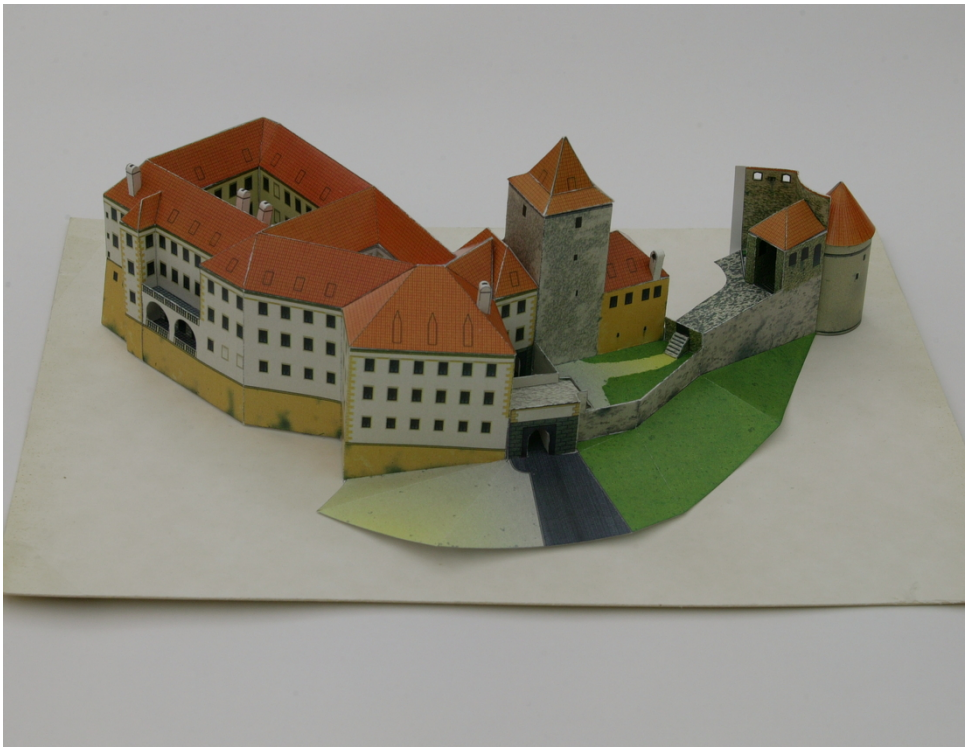
s uniformním konvolučním jádrem.[27] Uniformní konvoluční jádro je také separabilní a navíc lze hodnoty konvoluce vypočítat v lineárním čase[25], neboť pro sousední body i a $i+1$ při hodnotách konvolučního jádra w platí, že

$$\begin{aligned}
(f*h)(i) &= \sum_{m\in\mathcal{O}} h(i)f(i-m) \\
&= \sum_{i=i_1}^{i_2} wf(i-m) \\
&= wf(i_1-m) + \sum_{i=i_1+1}^{i_2} wf(i-m) \\
(f*h)(i+1) &= \sum_{m\in\mathcal{O}} h(i)f(i-m) \\
&= \sum_{i=i_1+1}^{i_2+1} wf(i-m) \\
&= wf(i_2-m) + \sum_{i=i_1+1}^{i_2} wf(i-m)
\end{aligned}$$

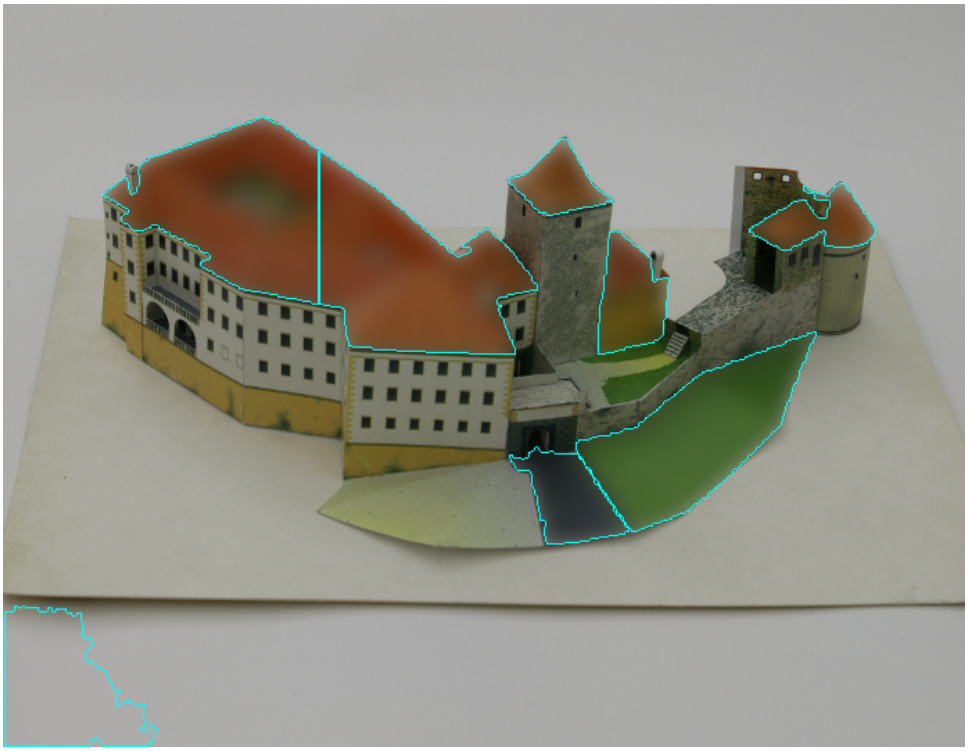
a hodnotu v bodě $i+1$ lze tedy v lineárním čase (pomocí jednoho přičtení a jednoho odečtení) získat z hodnoty v bode i jako

$$(f*h)(i+1) = (f*h)(i) - wf(i_1-m) + wf(i_2-m)$$

Časová složitost rozmazání je potom nezávislá na velikosti (uniformního) jádra (velikost jádra byla stanovena na sedm) a při trojnásobném konvolování, které bylo použito pro aproximaci, je rovna $O(6n)$.
Nahrazení segmentů v původním obraze odpovídajícími částmi rozmazaného obrazu je možné v jednom průchodu (pokud pixel patří segmentu je nahrazen pokud mu nepatří, pokračuje se dalším pixelem), což zvětšuje celkovou složitost na $O(7n)$. Hranice každého segmentu jsou pro snadnější rozpoznání dále označeny (tyrkysově), k čemuž je třeba spočítat řetězový kód každého segmentu. Při celkovém počtu segmentů k a počtu pixelů v segmentu N_s je tedy celková složitost rozmazání vyseparovaných částí ve vstupním obrázku rovna $O(7n + kN_s)$.
Porovnání původního obrázku a obrázku s rozmazanými částmi na místě segmentů je na Obr.36



(a) Původní obrázek



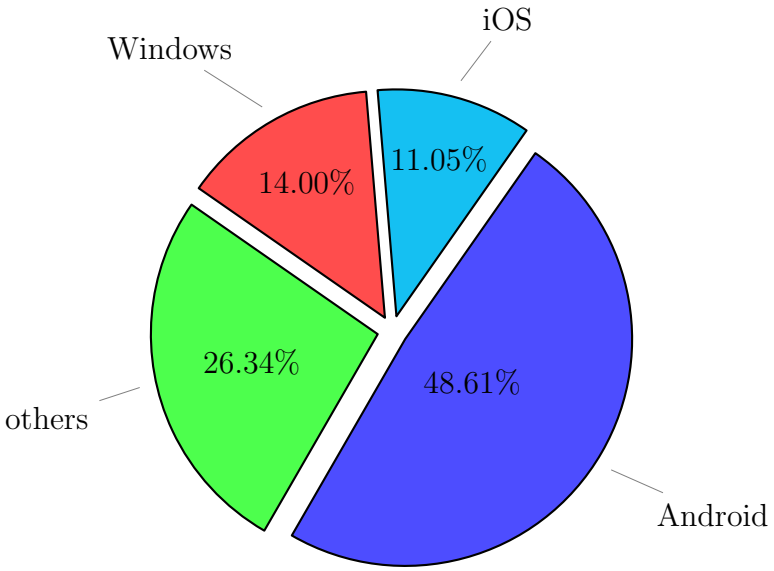
(b) Obrázek s rozmazanými částmi na místě segmentů

Obr. 36 Rozmazání vyseparovaných částí ve vstupním obrázku

Kapitola 4

Implementace

Výsledná aplikace implementující algoritmus popsaný v předchozí kapitole byla navržena a implementována pro operační systém Android[28][29]. Hlavním kritériem pro výběr tohoto operačního systémy bylo jeho značné rozšíření - v roce 2014 byl Android přítomen na 48.61% procentech (viz. Obr.37) zařízení (mobilní telefony, tablety a osobní počítače) a na rok 2015 je dokonce předpovídán nárůst na 58.90%[30]. Podíl na trhu mobilních zařízení je dokonce ještě mnohem vyšší - přibližně 80%[31].

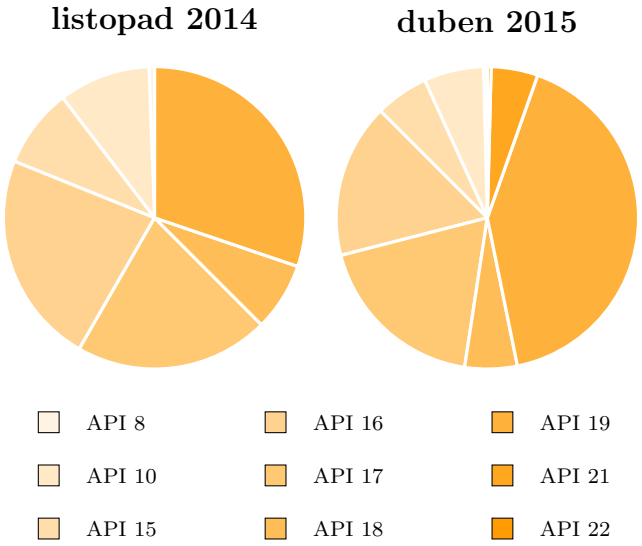


Obr. 37 Podíl operačních systémů na trhu v roce 2014

Společnost Google, která operační systém Android spravuje, vydává přibližně každých šest až devět měsíců inkrementální upgrade, a tak dochází ke značné

roztříštěnosti trhu, kdy nové verze operačního systému postupně vytlačují starší verze, viz Obr.38[32][33].

Verze	Kódové označení	API	Podíl v procentech	
			listopad 2014	duben 2015
2.2	Froyo	8	0.6	0.4
2.3.3 - 2.3.7	Gingerbread	10	9.8	6.4
4.0.3 - 4.0.4	Ice Cream Sandwich	15	8.5	5.7
4.1.x		16	22.8	16.5
4.2.x	Jelly Bean	17	20.8	18.6
4.3		18	7.3	5.6
4.4	KitKat	19	30.2	41.4
5.0	Lollipop	21	-	5.0
5.1		22	-	0.4



Obr. 38 Srovnání zastoupení verzí OS Android v listopadu 2014 a dubnu 2015

V době počátku vývoje (začátek podzimu 2014) například ještě neexistovala verze 5.0, která je nyní již na 5% zařízení. Aplikace je tedy dimenzována pro zařízení s verzí operačního systému Android 4.x (minimální je API verze 14, což odpovídá Android 4.0-4.0.2 Ice Cream Sandwich). Vývoj probíhal v programovacím jazyce Java.

4.1 Struktura aplikace

Základními stavebními kameny každé androidí aplikace jsou[34]:

- *Aktivity (Activities).* Uživatelské rozhraní aplikace je tvořeno aktivitami, které by se daly připodobnit oknům či dialogům běžné aplikace pro stolní počítače. Myšlenkou aplikace pro Android je velké množství jednoduchých aktivit, mezi kterými se v rámci aplikace přechází (je zde hojně využíváno tlačítko *Zpět* natolik běžné pro chytré telefony i tablety).
- *Služby (Services).* Aktivity mají zpravidla krátkou dobu života a jsou často ukončovány (například jinými aktivitami), proto existují služby, které jsou určeny k operacím probíhajícím „na pozadí“. Příkladem služby může být přehrávání hudby na pozadí - hudba hraje, i když aktivita, která přehrávání odstartovala již nemusí nutně existovat.
- *Poskytovatelé obsahu (Content providers).* Poskytovatelé obsahu umožňují sdílení dat mezi aplikacemi zařízení, k čemuž vývojový model pro Android přímo nabádá. Jedná se například o lokální databáze.
- *Záměry (Intents).* Záměry jsou systémové zprávy, pomocí kterých operační systém informuje aplikace o různých událostech (například datové přenosy). Navíc se záměry nepracuje pouze operační systém, ale i samotné aplikace - například vytvoření nové aktivity probíhá pomocí záměru.

Uživatelské rozhraní se skládá ze tří aktivit. Přesněji řečeno ze čtyř aktivit, jenže jednou z nich je galerie, kterou poskytuje přímo operační systém a její vzhled je tedy závislý na konkrétní verzi Androidu, proto zde nebude dále (příliš) diskutována.

(a) Schéma hlavní aktivity **(b)** Screenshot hlavní aktivity

Obr. 39 Hlavní aktivita *MainActivity*

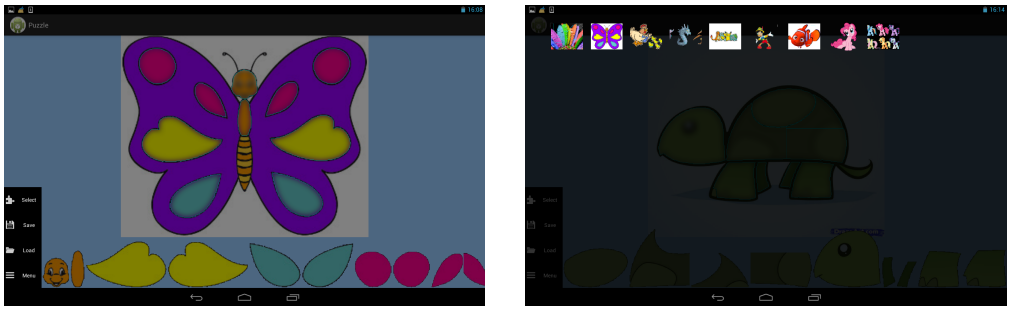
Hlavní aktivita aplikace *MainActivity* (viz. Obr.39) se skládá ze tří částí:

- Většinu aktivity zabírá plocha pro zobrazení vstupního obrázku, který může využít celou šířku a 80% výšky celé obrazovky. Po dokončení algoritmu je vstupní obrázek nahrazen svou rozmazanou variantou s vyznačenými hranicemi vyjmutých částí, viz Obr.36(b).
- V pravé dolní části je plocha pro jednotlivé dílky skládanky.
- Aplikace se ovládá z levé dolní části - možnost *Start*, po jejímž výběru dojde ke spuštění algoritmu z kapitoly 3, a možnost *Menu*, jež slouží k zobrazení dalších funkcionalit.

Menu aplikace, viz. Obr.40(a), je tvořeno čtyřmi tlačítky v levé dolní části, jejichž funkce je následující:

- *Select* - zvolení nového vstupního obrázku
- *Save* - uložení zpracovaného vstupního obrázku
- *Load* - načtení dříve uloženého vstupního obrázku
- *Menu* - opuštění menu

Zbytek obrazovky je „zastíněním“ hlavní aktivity (průhlednost 50%) a nemá žádné funkční využití.



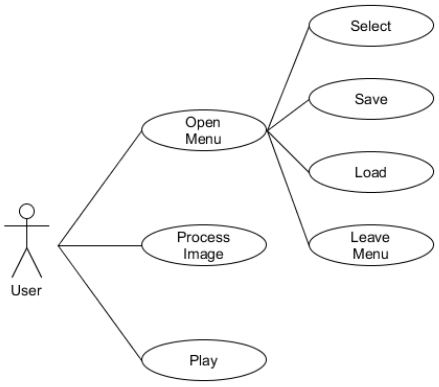
(a) *MenuActivity* **(b)** *LoadActivity*

Obr. 40 Ostatní aktivity

Poslední aktivitou je výběr uloženého obrázku při jeho načítání - *LoadActivity*, viz. Obr.40(b). Jedná se o dlaždicovou galerii s miniaturami uložených obrázků, která má stejně jako *MenuActivity* průhledné pozadí (celková průhlednost je snížena o dalších 30%, tj. na 20%).

4.2 Funkční vlastnosti aplikace

Případy užití aplikace jsou znázorněny na diagramu užití (use case diagram) na Obr.41.



Obr. 41 Diagram užití

Uživatel má možnost:

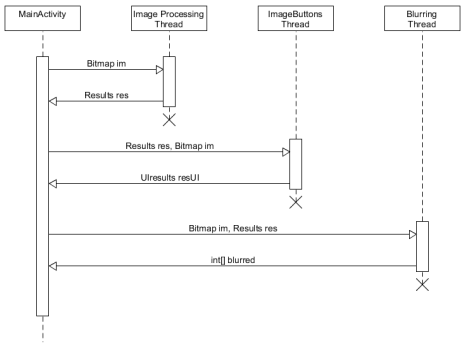
- Skládat stávající skládanku
- Zpracovat vybraný obrázek, tj. vytvořit z něj skládanku
- Vstoupit do menu a
  - Vybrat nový obrázek ke zpracování
  - Uložit stávající skládanku
  - Načíst dříve uloženou skládanku
  - Odejít z menu

4.2.1 Skládání skládanky

Samotné skládání, což je hlavním smyslem celé aplikace, je možné po zpracování vstupního obrázku nebo po načtení dříve uložených dat. Uživatel akcí „uchopit a táhnout“ („Drag & Drop“) umisťuje jednotlivé dílky na jejich příslušné pozice. Po úspěšném umístění je dílek odstraněn z hrací plochy, v obrázku je nahrazena rozmazaná část původní (tj. nerozmazanou) a zároveň dojde i k odstranění tyrkysových hranic. Po úspěšném umístění všech dílků vydá aplikace krátký oznamovací tón.

4.2.2 Zpracování obrázku

Všechny výpočetně složitější operace je nutno provádět mimo hlavní UI vlákno (v tomto případě *MainActivity*), neboť provádí-li toto vlákno jednu operaci delší dobu (několik málo vteřin), zobrazí android ANR chybovou hlášky „Aplikace neodpovídá“ („Application not responding“) a dojde k ukončení aplikace. Proto drtivá většina úkonů probíhá ve svých vlastních vláknech, které jsou v případě této aplikace spouštěny výhradně z hlavního UI vlákna. Výjimkou tomu není ani vytvoření skládanky z vybraného obrázku, jehož sekvenční diagram je na Obr.42.



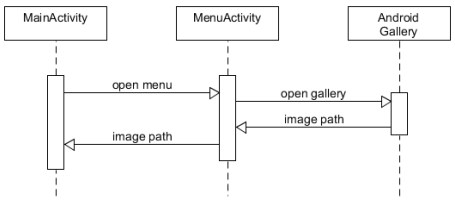
Obr. 42 Sekvenční diagram zpracování obrázku

Před samotným spuštěním výpočetních vláken jsou zablokována tlačítka *Start*, kterým je tento úkon spuštěn, a *Menu*. Nejprve je spuštěno vlákno *ImageProcessing*, které zpracuje obrázek (předaný vláknu ve formě bitmapy - objekt Bitmap) podle dříve popsaného algoritmu, nedojde však prozatím k rozmazání vyjmutých částí. *ImageProcessing* před svým ukončením odešle hlavnímu UI vláknu výsledky - seznam indexů každého výřezu, seznam indexů hranice každého výřezu a informace o umístění výřezu ve vstupním obrázku. Tyto výsledky společně se vstupním obrázkem jsou předány druhému vláknu (*ImageButtons*), které na jejich základě vytvoří UI prvky pro každý výřez a pošle je zpět hlavnímu vláknu, ve kterém jsou tyto prvky umístěny na obrazovku. Posledním vláknem je *Blurring*, ve kterém, jak již název napovídá, dojde na základě výsledků z *ImageProcessing* k rozmazání příslušných částí ve zpracovávaném obrázku.

Uživatel je v průběhu výpočtů informován pomocí progress baru o aktuálním stavu výpočtu. Po skončení všech tří vláken je zpřístupněna možnost *Menu* v *MainActivity*.

4.2.3 Výběr vstupního obrázku

Sekvenční diagram výběru obrázku ke zpracování je na Obr.43. Uživatel nejdříve stiskne tlačítko *Menu*, čímž dojde v aktivitě *MainActivity* k vytvoření nového záměru, který spustí aktivitu *MenuActivity*.

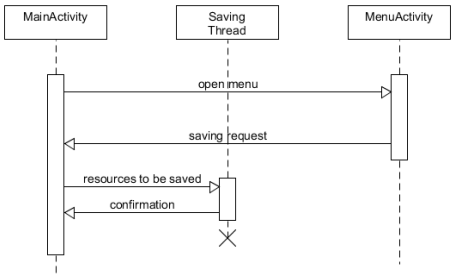


Obr. 43 Sekvenční diagram výběru obrázku

Výběrem *Select* v *MenuActivity* dojde, opět pomocí záměru, k otevření nativní androidí galerie, kde si uživatel může vybrat požadovaný vstup. Po vybrání putuje v obráceném pořadí cesta vstupního obrázku, který je v *MainActivity* umístěn na plochu jemu určenou (dojde k upravení velikosti obrázku tak, aby byl zachován poměr stran a zároveň aby byla využita co možná největší část této plochy). Po načtení nového vstupu je zpřístupněna možnost *Start*.

4.2.4 Uložení skládanky

Sekvenční diagram uložení skládanky je na Obr.44. Přístup do menu probíhá stejně jako při vybírání nového vstupu. Stisknutím tlačítka *Save* je hlavnímu UI vláknu oznámeno, že uživatel chce uložit stávající skládanku. Hlavní UI vlákno následně zablokuje tlačítka *Menu* a *Start* a spustí vlákno určené právě tomuto úkonu.



Obr. 44 Sekvenční diagram uložení skládanky

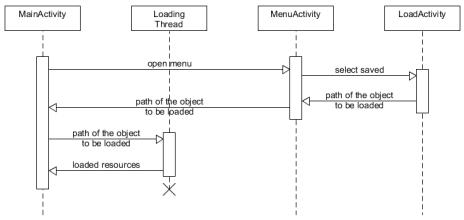
Každá skládanka je uložena ve formě šesti souborů s unikátním prefixem (současný čas v milisekundách) a následujícími koncovkami (první soubor má příponu .png, ostatní jsou bez přípony):

- PNG soubor bez koncovky - obrázek s rozmazanými a ohraničenými příslušnými částmi, slouží při načítání uložených skládanek
- bl - výčet RGBA hodnot pixelů obrázku (Android kóduje pro každý pixel tyto hodnoty do jednoho bajtu, každá složka zabírá 8 bitů), který má vyjmuté části rozmazané
- sg - výčet RGBA hodnot pixelů každého výřezu (jednotlivé segmenty jsou odděleny znakem x)
- pr - seznam UI parametrů každého výřezu (parametry jednotlivých segmentů jsou opět odděleny znakem x)
- in - bounding box každého výřezu
- pb - vzájemná poloha výřezů

Po načtení je opět zpřístupněno tlačítko *Menu* a *MainActivity* zobrazí potvrzovací dialog. Ukládání skládanky je možné pouze dokončení zpracování obrázku a nebo po načtení uložené skládanky; pokud je aktuální skládanka alespoň částečně složená, tlačítko *Save* je zablokováno.

4.2.5 Načtení uložené skládanky

Načtení uložené skládanky (Obr.45) je víceméně reverzní operace k jejímu uložení. Po výběru možnosti *Load* v *MenuActivity* dojde ke spuštění *LoadActivity*, kde si uživatel může vybrat jednu z uložených skládanek (každá skládanka je reprezentována zmenšeninou PNG souboru vytvořeného při jejím ukládání). Cesta uloženého souboru, konkrétně výše zmíněný prefix, je přes *MenuActivity* zaslána *MainActivity*, která spustí vlákno *Loading*. To na základě cesty načte informace z příslušných souborů a předá je zpět hlavnímu UI vláknu, které podle nich vytvoří v aplikaci skládanku.



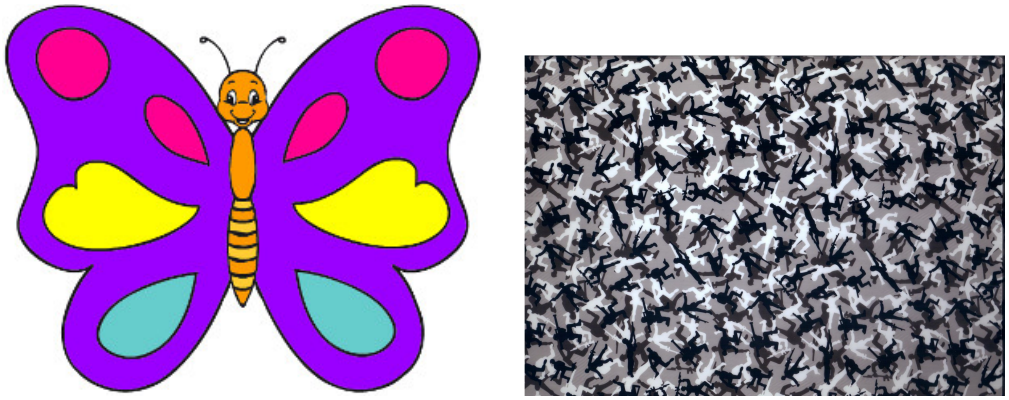
Obr. 45 Sekvenční diagram načtení uložené skládanky

Během načítání jsou opět zablokována obě tlačítka v *MainActivity* a po jeho dokončení je zpřístupněno tlačítko *Menu*.

Kapitola 5

Testování

Vzhledem k cílové skupině, pro kterou byla aplikace vyvíjena, byl algoritmus (konkrétně jeho parametry) navrhován pro specifický typ vstupních obrázků. Předpokládají se jednoduché obrázky, tvořené ze zřetelných a jednoduchých objektů (geometrických tvarů), viz. Obr.46. Algoritmus si přesto dokáže poradit s jakýmkoli vstupním obrázkem.



(a) Prototyp „dobrého“ vstupního obrázku

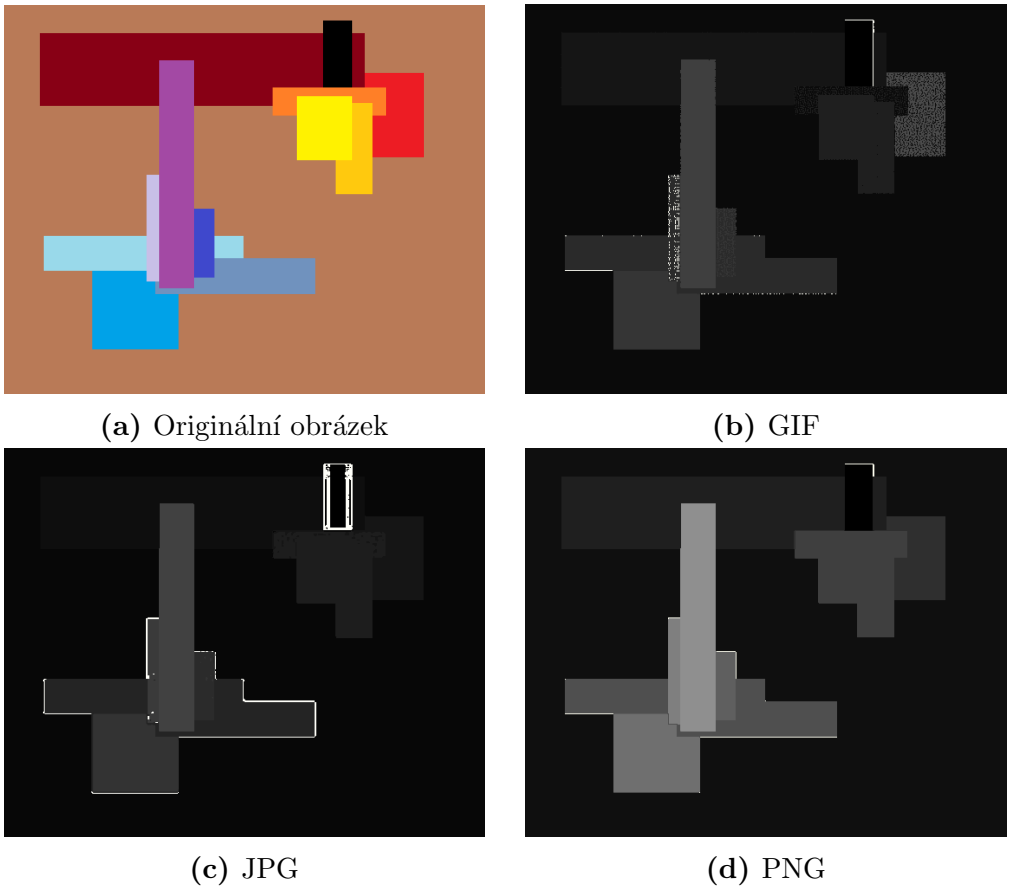
(b) Prototyp „špatného“ vstupního obrázku

Obr. 46 Prototypy vstupních obrázků

5.1 Okolnosti ovlivňující segmentaci obrazu

Výsledek algoritmu je závislý především na segmentaci obrazu, která je do značné míry ovlivněna formátem vstupního souboru. Jako experiment demonstrující tuto závislost byl vytvořen syntetický obrázek, Obr.47(a), který byl uložen ve všech třech běžných formátech, tj. PNG, JPG (JPEG) a GIF.

Každá varianta byla poté algoritmem zpracována a bylo dosaženo následujících závěrů.
PNG je bezeztrátový formát, což znamená, že komprese nesnižuje kvalitu obrázku[35]. Jak je patrné z Obr.47(d) (segmenty jako odstíny šedi), nedochází ke vzniku zrnitých oblastí na hranicích objektů popsaných v kapitole 3.3. Segmenty mají ostré hranice a jediné artefakty vznikající segmentací jsou úsečkovitého charakteru a není jich příliš. Segmentací vzniklo 17 segmentů, 7 z nich bylo hned po segmentaci odstraněno pro nedostatečnou velikost (výše zmíněné úsečkovité artefakty na hranicích objektů).



(a) Originální obrázek **(b)** GIF

(c) JPG **(d)** PNG

Obr. 47 Porovnání segmentace pro různé vstupní formáty

JPG soubory používají ztrátovou kompresi, takže dochází k patrné ztrátě kvality při přiblížení[35]. Segmentací JPG souborů, viz. Obr.47(c), vznikají zrnité oblasti na hranicích objektů přesně tak, jak bylo popsáno v kapitole 3.3, důsledkem čehož mohou mít segmenty členité hranice. Segmentací vzniklo 36 segmentů, 26 bylo pro nedostatečnou velikost záhy odstraněno (celkový počet dále zpracovávaných segmentů tedy zůstal stejný jako u PNG souboru).

GIF formát používá pouze paletu 256 barev. Barvy mimo tuto paletu jsou zaokrouhlovány tak, aby v ní ležely, čímž v obraze vznikají skvrnité oblasti[35]. Tyto oblasti jsou patrné i při segmentaci, Obr.47(b), kdy dochází k roztříštění objektů, které u předchozích dvou formátů zůstaly celistvé. Segmentací vzniklo 25 segmentů, ale 17 z nich bylo kvůli požadavkům na minimální velikost odstraněno a algoritmus tedy pokračoval s méně segmenty než tomu bylo o souborů PNG a JPG. Segmenty navíc mohou mít členité hranice.
Segmentace je mimo formátu vstupního souboru ovlivněna třemi parametry - počtem binů v trojrozměrném histogramu. Volbou poměru binů 15:7:7 (pro složky H:S:V) ovšem dojde k mírnému podsegmentování, což je patrné z Obr.48. Konkrétně nejsou rozeznány podobné odstíny, například červená a oranžová nebo tmavě modrá a fialová. Zvýšením počtu binů by sice došlo k odstranění tohoto nechtěného efektu, docházelo by ovšem naopak k přesegmentování obrazu.

(a) Originální obrázek **(b)** Segmentace

Obr. 48 Rozlišování odstínů při segmentaci

Poměr 15:7:7 byl tedy zvolen jako přijatelný kompromis. Navíc se počítalo s jednoduchostí vstupních obrázků, konkrétně se při obrázcích složených z jednoduchých geometrických tvarů neočekávala přílišná barevná shoda sousedních tvarů.

5.2 Testování s uživateli

Pro testování s uživateli byly vytvořeny 3 sady vstupních obrázků:

- Set I. 30 jednoduchých obrázku často tvořených pouze jednoduchými geometrickými tvary. Obrázky obsahují jasně identifikovatelné a rozeznatelné objekty a mají jednoduché (jednobarevné) pozadí.
- Set II. 51 mírně složitějších obrázků - identifikace objektů je složitější než u předchozí sady a pozadí může být složitější.
- Set III. 17 složitých obrázků - identifikace objektů není zřejmá, velmi často komplexní pozadí.

Testování se účastnilo celkem 5 osob; každé z nich byly ukazovány vstupní obrázky a výsledná skládanka vytvořená aplikací. Tester měl subjektivně posoudit vizuální přitažlivost vyseparovaných dílků a jejich korespondenci s objekty v obraze podle stupnice:

1 - velmi špatné

2 - špatné

3 - uspokojivé

4 - dobré

5 - velmi dobré

Výsledky testování, viz. Obr.49, jsou v souladu s požadovanými cíli. Set I, tedy sada obrázků, s kterými je vzhledem k cílové skupině aplikace počítáno jako s žádoucími vstupy, je hodnocen pozitivně (nejlépe hodnocený je obrázek Obr.46(a)). Naopak Set III, který obsahuje například fotografie krajiny či staveb, se setkal spíše s negativními ohlasy (nejhůře hodnocený je obrázek Obr.46(b)).

	Uživatel					Průměr
	01	02	03	04	05	
I	3.83	4.30	3.93	3.33	3.70	3.82
Set II	3.39	3.63	3.33	2.86	3.35	3.31
III	2.23	3.06	2.06	2.12	2.52	2.40

Obr. 49 Hodnocení uživatelů

Kapitola 6

Závěr

Byla vytvořena funkční aplikace pro operační systém Android splňující základní požadavky - automatickou tvorbu hry typu puzzle z libovolného vstupního obrázku na základě segmentace obrazu. Aplikace umožňuje uživateli, mimo výběru vstupního obrázku libovolného formátu a vlastního hraní, uložení vytvořené skládanky a její opětovné načtení.
Přesto existují oblasti, které by bylo možné v budoucím vývoji zlepšit. Co se týká algoritmu pro tvorbu skládanky, nebylo by od věci zvážit i jiná kritéria pro výběr žádoucích dílků než jejich kompaktnost, případně kritéria nějak zkombinovat. Dále by stály za zvážení další možnosti při zobrazení vyjmutých částí ve vstupním obrázku, např. efekt vtlačení.
Také aplikace samotná by mohla nabízet další volby. Při tvorbě velkého počtu dílků (může jich být až několik desítek na jeden obrázek) by přišla vhod možnost pro přepínání počtu zobrazených dílků (například 1-5, 5-10, 10+). Další funkcí, která by mohla být dále implementována, je mazání uložených skládanek přímo z aplikace, prozatím je uloženou skládanku možné odstranit pouze ručním smazáním příslušných souborů ve složce aplikace.

Literatura

[1] *Google Play* [online]. [cit. 2015-05-04]. Dostupné z: https://play.google.com/store

[2] Google Play. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-05-04]. Dostupné z: http://cs.wikipedia.org/wiki/Google_Play

[3] Animal Jigsaw Puzzles For Kids. *Google Play* [online]. [cit. 2015-05-04]. Dostupné z: https://play.google.com/store/apps/details?id=com.tula.kidjigsaws

[4] Toddlers Puzzle Woozzle. *Google Play* [online]. [cit. 2015-05-04]. Dostupné z: https://play.google.com/store/apps/details?id=com.woozle

[5] ŠONKA, Milan, Václav HLAVÁČ a Roger BOYLE. *Image processing, analysis, and machine vision*. 3rd ed. Toronto: Thomson, 2008, xxv, 829 s. ISBN 978-0-495-08252-1.

[6] Lectures on Image Processing. PETERS II, Richard Alan. [online]. [cit. 2015-04-16]. Dostupné z: https://archive.org/details/Lectures_on_Image_Processing

[7] Zoom An Image With Different Interpolation Types. [online]. [cit. 2015-04-16]. Dostupné z: http://www.codeproject.com/Articles/236394/Bi-Cubic-and-Bi-Linear-Interpolation-with-GLSL

[8] PRATT, William K. *Digital image processing: PIKS Scientific inside*. 4th ed. Hoboken.: Wiley-Interscience, c2007, xix, 782 s. ISBN 978-0-471-76777-0.

[9] MITCHELL, Don P. a Arun N. NETRAVALI. Reconstruction filters in computer-graphics. *ACM SIGGRAPH Computer Graphics*. 1988, vol. 22, issue 4, s. 221-228. DOI: 10.1145/378456.378514.

[10] HARALICK, Robert M. a Linda G. SHAPIRO. Image segmentation techniques. *Computer Vision, Graphics, and Image Processing*. 1985, vol. 29, issue 1, s. 100-132. DOI: 10.1016/s0734-189x(85)90153-7.

[11] The Berkeley Segmentation Dataset and Benchmark. ARBELAEZ, Pablo, Charless FOWLKES a David MARTIN. [online]. [cit. 2015-04-17]. Dostupné z: http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/

[12] ROTHER, Carsten, Vladimir KOLMOGOROV a Andrew BLAKE. "GrabCut". *ACM Transactions on Graphics*. 2004-08-01, vol. 23, issue 3, s. 309-. DOI: 10.1145/1015706.1015720. Dostupné z: http://portal.acm.org/citation.cfm?doid=1015706.1015720

[13] PATIL, R. V. a K. C. JONDHALE. Edge based technique to estimate number of clusters in k-means color image segmentation. *2010 3rd International Conference on Computer Science and Information Technology*. 2010. DOI: 10.1109/iccsit.2010.5563647.

[14] TURI, Rose a Siddheswar RAY. Determination of number of clusters in K-means clustering and application in colour image segmentation. In: PAL, Nikhil R, Arun K DE a Jyotirmay DAS. *Advances in pattern recognition and digital techniques*. New Delhi: Narosa Pub. House, c2000, s. 137-143. ISBN 8173193479.

[15] OHASHI, Takumi, Zaher AGHBARI a Akifumi MAKINOUCHI. Hill-climbing Algorithm for Efficient Color-based Image Segmentation. In: HAMZA, Ed.: M. H. *Proceedings of the IASTED International Conference on Signal Processing, Pattern Recognition, and Applications: June 30 - July 2, 2003, Rhodes, Greece*. Anaheim [u.a.]: Acta Press, 2003, s. 59-97. ISBN 0889863636.

[16] RGB to HSV conersion. *Online Reference & Tools* [online]. 2014 [cit. 2015-04-18]. Dostupné z: http://www.rapidtables.com/convert/color/rgb-to-hsv.htm

[17] LI, Wenwen, Michael F. GOODCHILD a Richard CHURCH. An efficient measure of compactness for two-dimensional shapes and its application in regionalization problems. International Journal of Geographical Information Science. 2013, vol. 27, issue 6, s. 1227-1250. DOI: 10.1080/13658816.2012.752093.

[18] FRIEDENBERG, Jay. Aesthetic judgment of triangular shape: compactness and not the golden ratio determines perceived attractiveness. *I-Perception* 2012, vol. 3, issue 3, s. 163-175. DOI: 10.1068/i0484. Dostupné z: http://i-perception.perceptionweb.com/journal/I/article/i0484

[19] FREEMAN, Herbert. On the Encoding of Arbitrary Geometric Configurations. *IEEE Transactions on Electronic Computers*. 1961, EC-10, issue 2, s. 260-268. DOI: 10.1109/TEC.1961.5219197. Dostupné z: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5219197

[20] Measuring boundary length. *Cris's Image Analysis Blog* [online]. [cit. 2015-04-25]. Dostupné z: http://www.cb.uu.se/ cris/blog/index.php/archives/310

[21] How to obtain the chain code. *Cris's Image Analysis Blog* [online]. [cit. 2015-04-25]. Dostupné z: http://www.cb.uu.se/ cris/blog/index.php/archives/324

[22] PROFFITT, D. a D. ROSEN. Metrication errors and coding efficiency of chain-encoding schemes for the representation of lines and edges. *Computer Graphics and Image Processing*. 1979, vol. 10, issue 4, s. 318-332. DOI: 10.1016/S0146-664X(79)80041-6. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/S0146664X79800416

[23] VOSSEPOEL, A.M. a A.W.M. SMEULDERS. Vector code probability and metrication error in the representation of straight lines of finite length. *Computer Graphics and Image Processing*. 1982, vol. 20, issue 4, s. 347-364. DOI: 10.1016/0146-664X(82)90057-0. Dostupné z: http://linkinghub.elsevier.com/retrieve/pii/0146664X82900570

[24] Gaussian blur. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-29]. Dostupné z: http://en.wikipedia.org/wiki/Gaussian_blur

[25] Fastest Gaussian Blur (in linear time). KUCKIR, Ivan. *Algorithms and Stuff* [online]. [cit. 2015-04-29]. Dostupné z: http://blog.ivank.net/fastest-gaussian-blur.html

[26] Separable convolution. *MATLAB Central: Steve on Image Processing* [online]. 2006 [cit. 2015-04-29]. Dostupné z: http://blogs.mathworks.com/steve/?p=78?s_cid=srchtitle

[27] Gaussian filtering. *Cris's Image Analysis Blog* [online]. [cit. 2015-04-29]. Dostupné z: http://www.cb.uu.se/ cris/blog/index.php/archives/22

[28] *Android* [online]. [cit. 2015-05-02]. Dostupné z: https://www.android.com/

[29] Android (operating system). In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-05-02]. Dostupné z: http://en.wikipedia.org/wiki/Android_%28operating_system%29

[30] Gartner Says Tablet Sales Continue to Be Slow in 2015. *Gartner Inc.: Technology Research* [online]. 2015 [cit. 2015-05-02]. Dostupné z: http://www.gartner.com/newsroom/id/2954317

[31] Smartphone OS Market Share, Q4 2014. *IDC: The premier global market intelligence firm* [online]. 2015 [cit. 2015-05-02]. Dostupné z: http://www.idc.com/prodserv/smartphone-os-market-share.jsp

[32] Android Distribution Updated for April 2015 – Lollipop Jumps to 5.4%. textitDroid Life [online]. [cit. 2015-05-02]. Dostupné z: http://www.droid-life.com/2015/04/08/android-distribution-updated-for-april-2015-lollipop-jumps-to-5-4/

[33] Android Distribution Updated for November 2014, Kit Kat Hits 30%. *Droid Life* [online]. [cit. 2015-05-02]. Dostupné z: http://www.droid-life.com/2014/11/03/android-distribution-updated-for-october-2014-kit-kat-hits-30/

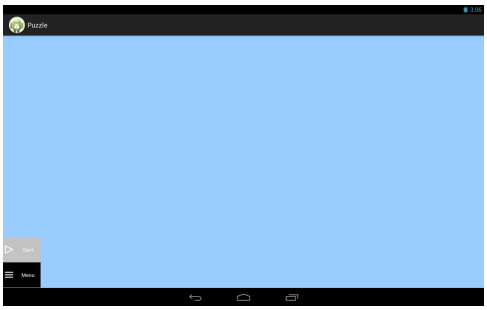
[34] ALLEN, Grant. *Beginning Android 4*. Apress: distributed by Springer Science Business Media, c2012, xx, 582 pages. ISBN 14-302-3984-0.

[35] GIF vs. JPG vs. PNG: What's the Difference?. *Digital Trends: Technology News and Product Reviews* [online]. [cit. 2015-05-04]. Dostupné z: http://www.digitaltrends.com/computing/whats-the-difference-between-a-gif-a-jpg-and-a-png-file/

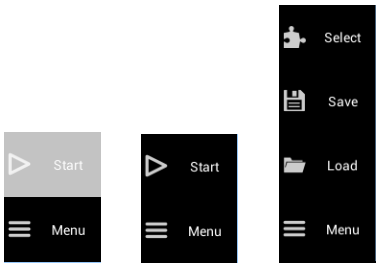
Příloha A

Ovládání aplikace

Výchozí stav aplikace je na Obr.50. Aplikace je ovládána dvěma tlačítky v levém dolním rohu hlavního okna, viz Obr.51(b) - pokud má tlačítko šedé pozadí (viz. tlačítko *Start* na Obr.51(a)), je zablokované, pokud má černé pozadí, je aktivní (uživatel jej může použít).



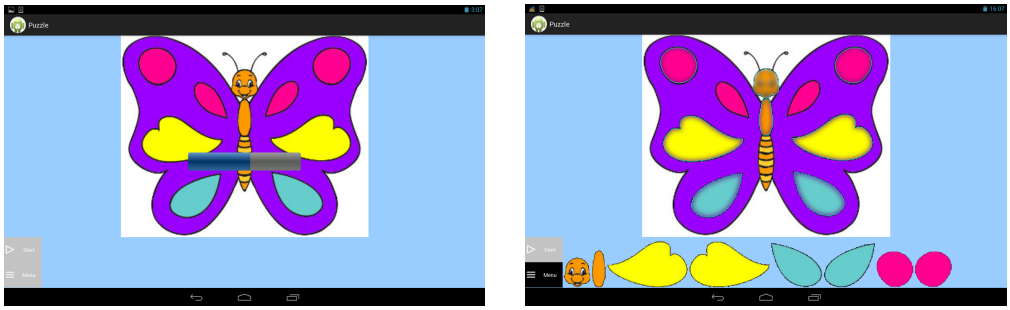
Obr. 50 Výchozí obrazovka



Obr. 51 Ovládání aplikace

Ve výchozím stavu je přístupná jen možnost *Menu*, po jejímž zvolení se zobrazí menu z Obr.51(c).

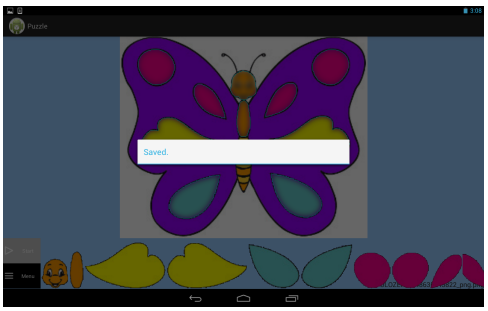
Pro vybrání nového obrázku ke zpracování je třeba otevřít menu a zvolit *Select* (ikona s puzzlem), které otevře galerii (její vzhled je závislý na verzi operačního systému) pro výběr vstupního obrázku.
Po výběru nového vstupu je zpřístupněna možnost *Start* v hlavním okně, viz. 51(b), po jejíž volbě dojde ke zpracování obrázku. Během výpočtu jsou zablokovány možnosti *Start* a *Menu* a uživatel je informován o průběhu pomocí progress baru, viz. Obr.52(a). Po dokončení zpracovávání zmizí progress bar a na pozici vedle ovládacích prvků jsou vyskládány vytvořené dílky, viz. Obr.52(b).



(a) **(b)**

Obr. 52 Průběh zpracování

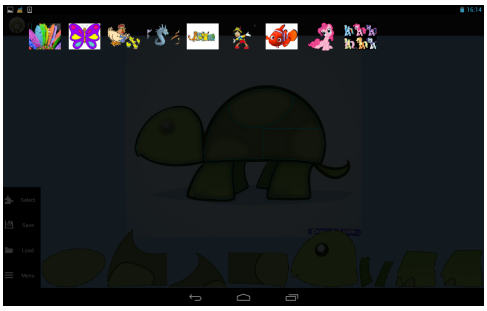
Uložení vytvořené skládanky se provádí z menu pomocí tlačítka *Save*, viz. 51(c), které je aktivní pouze po dokončení zpracování vstupu a nebo po načtení uložené skládanky. Během ukládání je uživatel přenesen zpět na hlavní obrazovku a dojde ke krátkodobému zablokování možností *Start* a *Menu*. Uživatel je posléze informován o úspěšném provedení uložení, viz. Obr.53, načež je opět zpřístupněna možnost *Menu*.



Obr. 53 Potvrzení uložení

Načtení dříve uložené skládanky je možné pokaždé, pokud je přístupna možnost *Menu*. Po zvolení možnosti *Load* v menu je zobrazena dlaždicová galerie s miniaturami uložených skládanek, viz. Obr.54. Po výběru uloženého objektu

je uživatel přenesen zpět do hlavního okna, kam je načtena i požadovaná skládanka.



Obr. 54 Dlaždicová galerie pro načítání

Tlačítko *Menu*, viz. Obr.51(b) a Obr.51(c), se vyskytuje jak na hlavní obrazovce, tak v menu aplikace a slouží pro přepínání mezi těmito dvěma.
Vlastní hraní hry probíhá intuitivně pomocí operace „uchopit a táhnout“ („Drag & Drop“), kdy uživatel umisťuje jednotlivé dílky na jejich původní pozice. Po umístění všech dílků je spuštěn vítězný tón.

Příloha B

Seznam použitých zkratek

HSV	Hue Saturation Value
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
px	pixel
IPQ	Isoperimetric Quotient
DCM	Digital Compactness Measure
NDC	Normalized Discrete Compactness
CoG	Center of Gravuty
API	Application Programming Interface
ANR	Application Not Responding
UI	User Interface
PNG	Portable Network Graphics
JPG, JPEG	Joint Photographic Experts Group
GIF	Graphics Interchange Format

Příloha C

Obsah přiloženého DVD

```
DVD
|   obsah.txt                   obsah DVD
|
+---App                         složka s aplikací
|       Puzzle.apk              instalační soubor aplikace
|
+---src                         složka se zdrojovými kódy aplikace
|   |
|   \---doc                     dokumentace zdrojového kódu aplikace
|
+---text                        složka s textem diplomové práce
|       DP_pruner.pdf           vlastní text diplomové práce
|
\---Data                        testovací vstupní obrázky
    |
    +---01                      1. set obrázků (jednoduché)
    |
    +---02                      2. set obrázků (složitější)
    |
    \---03                      3. set obrázků (složité)
```

Obr. 55 Obsah přiloženého DVD