UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

GENEROVANIE VZOROV NA VYŠÍVANIE

Bakalárska práca

2014

Anna Dresslerová

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

GENEROVANIE VZOROV NA VYŠÍVANIE

Bakalárska práca

Študijný program:	Informatika
Študijný odbor:	2508 Informatika
Školiace pracovisko:	Katedra informatiky
Vedúci práce:	RNDr. Michal Forišek, PhD.

Bratislava, 2014

Anna Dresslerová

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Anna Dresslerová
Študijný program: informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
Študijný odbor: 9.2.1. informatika
Typ záverečnej práce: bakalárska
Jazyk záverečnej práce: slovenský

Názov: Generovanie vzorov na vyšívanie

Cieľ: Hlavným cieľom práce je vytvorenie aplikácie, ktorá z danej bitmapovej predlohy vygeneruje vzor pre vyšívanie. Súčasťou práce je preskúmanie existujúcich a samostatný vývoj nových prístupov k riešeniu súvisiacich algoritmických problémov. Medzi konkrétne problémy, ktoré práca musí riešiť, patrí mapovanie celého spektra farieb na vopred určenú sadu dostupných farieb vlny; a taktiež samotná realizácia zmeny rozlíšenia obrázku, pri ktorej bude potrebné preskúmať nakoľko vhodné sú pre tento špecifický problém rôzne formy ditheringu. Textová časť práce by mala obsahovať rozbor tejto problematiky.

Vedúci: RNDr. Michal Forišek, PhD.
Katedra: FMFI.KI - Katedra informatiky
Vedúci katedry: doc. RNDr. Daniel Olejár, PhD.

Dátum zadania: 22.10.2013

Dátum schválenia: 23.10.2013

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.....
študent

.....
vedúci práce

Pod'akovanie

Chcela by som sa pod'akovať vedúcemu tejto práce RNDr. Michalovi Foriškovi, PhD., za usmernenia pri písaní práce. Taktiež by som sa chcela pod'akovať Lucii Petríkovej za promptné nakreslenie testovacej bitmapy a neposlednom rade mojej rodine za neustálu podporu.

Abstrakt

V práci skúmame existujúce a navrhujeme nové riešenia problémov súvisiacich s generovaním predlohy na krížikové vyšívanie z bitmapy. Súvisiace problémy sú výber vhodnej palety farieb dostupných vyšívacích priadzí, zmena veľkosti vstupnej bitmapy na rozmery výšivky a určenie jednotlivým krížikom, akou priadzou majú byť vyšité. Pri poslednom probléme skúmame vhodnosť rôznych typov ditheringu. K práci je taktiež priložená aplikácia, v ktorej sú naše riešenia implementované a tiež obsahuje používateľské rozhranie, ktoré okrem iného umožňuje vygenerovanú predlohu editovať, ukladať, otvárať a exportovať do PDF.

Kľúčové slová: krížikové vyšívanie, digitalne spracovanie obrazu, dithering

Abstract

This work is dedicated to examining existing and developing new solutions to problems connected with generating patterns for cross-stitch embroidering from a bitmap. These problems encompass choosing the correct colour palette of available embroidering yarn, adjusting the size of the input bitmap to the size of the embroidery and assigning the correct yarn type to the respective stitches. The suitability of different types of dithering is researched into by the last problem. The work includes an application with our implemented solutions and a user interface that enables editing, saving, opening and exporting to PDF format of the generated pattern.

Keywords: cross-stitching, digital image processing, dithering

Obsah

Úvod

Cieľom tejto práce je navrhnúť a následne implementovať riešenia pre problém generovania predlohy na krížikové vyšívanie. Vstupom tohto problém je, okrem iného, obrázok vo forme bitmapy. Výšivka, ktorá sa vyšije podľa vygenerovanej predlohy, by sa mala či najviac podobať na vstupný obrázok. Okrem bitmapy je ďalej vstupom paleta farieb daná vybratými priadzami na vyšívanie, maximálny počet vyšívacích priadzí použitých vo výšivke a rozmery výšivky. Problém sme si rozdelili na tri samostatné podproblémy:

1. Problém vyberu čo najlepšej množiny vyšívacích priadzí, pričom počet priadzí nemôže presiahnuť maximálny zadaný počet.

2. Problém zmeny veľkosti bitmapy v pixeloch na rozmery výšivky v krížikoch.

3. Problém priradenia vyšívacích priadzí krížikom výšivky.

Podobným problémom sa zaoberá veľa ľudí. Svedčí o tom množstvo miniaplikácii, ktoré sa dajú nájsť na internete. Medzi týmito aplikáciami sme našli jeden, voľne dostupný a profesionálny softvér, ktorý dokonca ponúka možnosť použitia tzv. ditheringu. Dithering aj my v práci skúmame ako jedno z riešení tretieho problému. Tento softvér sa volá Stitch Art Easy! 4.0 a môžete ho nájsť na stránke `http://stitcharteasy.com/`.

Prvú kapitolu venujeme základným pojmom, ktoré budeme v práci používať. Ďalej každému z troch vyššie spomenutých problémov venujeme jednu kapitolu, v ktorej popíšeme niekoľko jeho riešení. Piatu kapitolu venujeme implementácii. V závere porovnáme výstupy našej aplikácie a programu Stitch Art Easy! 4.0.

Kapitola 1

Základné pojmy

1.1 Práca s farbami

V práci prakticky neustále pracujeme s farbami. Na prácu s nimi používame tzv. RGB farebný model. V tomto modeli sa farba rozkladá na tri základné farby, a to: červenú (Red), zelenú (Green) a modrú (Blue). Pre každú základnú farbu máme vyhradených 8 bitov. Každá zo základných zložiek teda môže nadobúdať hodnoty od 0 po 255.

Definícia. *Farbou* nazývame trojicu prirodzených čísel (r, g, b) v rozsahu od 0 po 255. Niekedy ju budeme považovať za trojrozmerný vektor. Množinu všetkých farieb budeme označovať $\mathbb{F}$.

Často budeme potrebovať zistiť najbližšiu farbu k nejakej inej farbe, alebo jednoducho zistiť vzdialenosť. Budeme na to používať eulerovskú vzdialenosť.

Definícia. Nech máme dané dve farby v RGB modeli: $c_1 = (r_1, g_1, b_1)$, $c_2 = (r_2, g_2, b_2)$. Potom vzdialenosť $d(c_1, c_2)$ týchto farieb je:

$$d(c_1, c_2) = \sqrt{(r_2 - r_1)^2 + (g_2 - g_1)^2 + (b_2 - b_1)^2} \tag{1.1}$$

Táto vzdialenosť nie je najlepšia možná, lebo neberie do úvahy vlastnosti ľudského oka. Pre naše potreby však stačí. Navyše nie je problém neskôr vzdialenosť definovať inak. Všetky algoritmy popísané v nasledujúcich kapitolách nie sú zavyslé na tejto definícii vzdialenosti.

Ako zadanie našej práce napovedá, budeme taktiež veľa pracovať s bitmapovým vstupom.

Definícia. *Bitmapa B* je matica bodov, ktoré sa nazývajú pixely. Pixely bitmapy B budeme označovať súradnicami $(x, y)_B$. Každý pixel $(x, y)_B$ bitmapy má farbu, ktorú budeme označovať $p_B(x, y)$

1.2 Terminológia pre vyšívanie

Na to, aby sme mohli vyšívať, potrebujeme v zásade tri veci. Ihlu, tkaninu na ktorú vyšívame a vyšívacie priadze. Pre našu prácu je zaujímavé, že tkanina vytvára švorcovú sieť, pričom dierky sú v uzlových bodoch. Dôležitejšie sú však vyšívacie priadze, lebo tie tvoria farebnosť výšivky. Na trhu je niekoľko značiek vyšívacích priadzí. Každá značka má svoju farebnicu, ktorá sa snaží pokryť celé farebné spektrum. Bohužiaľ ho nepokrýva rovnomerne. Lepšie značky vyšívacích priadzí ponúkajú vo svojej farbenici niekoľko stoviek farieb. Napr. značka Anchor ich má 438. Farbám farebnice sú priradené čísla, podľa ktorých sa vyšívacie priadze dajú neskôr kúpiť. Z toho vyplýva, že farba vyšívacej priadze je jednoznačne určená značkou a jej číslom vo farebnici.

Definícia. *Vyšívacia priadza* je dvojica (c, z), kde c je číslo vo farebnici vyšívacích priadzí značky z. Množinu všetkých dostupných vyšívacích priadzí budeme označovať $\mathbb{T}$.

Definícia. *Farebnica* je zobrazenie $F : \mathbb{T} \to \mathbb{F}$.

Definícia. *Paleta* je množina vyšívacích priadzí.

Často budeme hľadať vyšívaciu priadzu z palety, ktorej farba je najbližšia zadanej farbe. Zavedieme si na to špeciálne označenie:

Definícia. Nech P je paleta a c je farba. Farebne najbližia vyšívacia priadza z palety P k farbe c je $\Phi_P(c) = t_0$. Pričom

$$\forall t \in P : d(c, F(t_0)) \leq d(c, F(t)) \tag{1.2}$$

Táto farebne najbližšia priadza nemusí byť iba jedna, ale v tom prípade si vyberieme ľubovoľnú z nich.

Ďalším dôležitým pojmom, ktorý budeme používať, je predloha. Predlohy, ktoré nájdeme v obchodoch, obsahujú štvorcovú sieť, pričom v každom štvorci je symbol. Stvorcová sieť znázorňuje tkaninu a priesečníky čiar sú dierky v tkanine. Každý štvorček predstavuje jeden krížik výšivky a symbol v ňom priadzu, ktorou sa má zodpovedajúci krížik vyšiť.

K tejto štvorcovej sieti je ďalej priložená legenda, kde každému symbolu v sieti je priradená vyšívacia priadza tak, že je uvedená jej značka a číslo vo farebnici. Ďalej je ku každej priadzi uvedené, koľko pradien bude treba na vyšitie výšivky. Jedno pradeno má štandardne osem metrov. Počet pradien sa zvyčejne určuje podľa počtu krížikov, ktoré sa danou vyšívacou priadzou majú vyšiť. My sme experimentálne zistili, že jedným metrom vyšívacej priadze sa dá vyšiť 170 krížikov. Je však potrebné nechať rezervu, lebo krížiky nemusia byť blízko pri sebe a vtedy je spotreba vyšívacej priadze vyššia. Určili sme, že počet pradien vypočítame podľa vzorca $\lceil (p + 1)/1000 \rceil$, kde p je počet krížikov. Tento odhad by sa dal

ešte vylepšiť. Do počítania odhadu by sa mohli ešte zapojiť aj vzdialenosti medzi krížikmi vyšitými danou vyšívacou priadzou. Pre malé výšivky však náš odhad stačí.

Keďže tento model sa štandardne používa všade na svete, rozhodli sme sa ho použiť aj my. Vzhľadom na pohodlnosť manipulácie s ním, sme však v definícii urobili mierne zmeny.

Definícia. Množinu všetkých symbolov označíme $\mathbb{S}$.

Definícia. *Predloha* je dvojica $R = (M, S)$, kde M je matica typu $m \times n$, pričom jej prvky sú z množiny $\mathbb{T} \cup \{\varepsilon\}$. Prvok matice M na súradniciach $(x, y) \in \{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\}$ budeme označovať $M(x, y)$. Nech T je množina priadzí v matici M: $T = \{t \in \mathbb{T} \mid \exists (x, y) \in \{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\} : M(x, y) = t\}$. S je injektívne zobrazenie $S : T to \mathbb{S}$.

Ako ste si mohli všimnúť, do matice sme nedali symboly, ale priamo priadze. Ak je nejaký prvok matice ε, to znamená, že miesto, ktoré zodpovedá tomuto prvku vo výšivke, zostane prázdne. Počty krížikov máme zaznamenané priamo v matici. Z takto definovanej predlohy sa dá ľahko vyrobiť jej tlačená verzia.

1.2.1 Trieda `ColorPalette`

Pre štruktúru *paleta*, ktorú sme si definovali vyššie sme implementovali samostatnú triedu. Okrem toho, že táto trieda uchováva vyšívacie priadze (v zmysle vyššie popísanej definície) a k nim prislúchajúce farby, má aj nasledujúce funkcie:

- `addColorsFromFile(paletteName)` - pridá paletu danej značky.
- `addColor(id,color)` - pridá novú vyšívaciu priadzu `id` s farbou `color`. Ak sa v nej taká vyšívacia priadza nachádza s inou farbou (čo by sa nemalo stať), tak starú farbu nahradí novou.
- `removeColor(id)` - odstráni priadzu `id`. Ak tam taká nie je, tak sa nič nestane.
- `clear()` - vyprázdni paletu.
- `getIDsOfColors()` - vráti množinu všetkých priadzí, ktoré sú v palete.
- `colorIsInPalette(id)` - vráti `true`, ak sa v palete nachádza vyšívacia priadza `id`. V opačnom prípade vráti `false`.
- `getColor(id)` - vráti farbu priadze `id`.
- `getNearestIDToColor(color)` - vráti priadzu, ktorej farba je najbližšia k farbe `color`.
- `findDistance(id1,id2)` - vráti vzdialenosť farieb vyšívacích priadzí `id1` a `id2`.

1.2.2 Trieda `Pattern`

Aj pre štruktúru *predloha* sme implementovali samostatnú triedu. Uchováva v sebe maticu a zobrazenie S. Navyše pre jednoduchosť si priebežne aktualizuje počty výskytov všetkých priadzí v matici. Namiesto ε v programe používame hodnotu `None`. Navyše má táto trieda tieto funkcie:

- `getHeight()` - vráti výšku predlohy v krížikoch
- `getWidth()` - vráti šírku predlohy v krížikoch
- `getSquare(x,y)` - vráti priadzu alebo hodnotu `None`, ktorá je v matici na zadanej pozícií. Ak súradnice nie sú korektné, vráti `(-1)`.
- `getColorSymbol(id)` - vráti symbol pre vyšívaciu priadzu `id`.
- `getColorCount(id)` - vráti počet výskytov vyčívacej priadze `id` v matici.
- `getColors()` - vráti množinu priadzí v matici.
- `getColorToSym()` - vráti celý prevodník (zobrazenie) medzi priadzami a symbolmi.
- `setSquare(x,y,id)` - do matice vloží na zadanú pozíciu priadzu `id`. Ak zadaná pozícia neexistuje, nič sa nestane. Ak sa týmto úkonom pridala do predlohy nová vyšívacia priadza, tak jej priradí voľný symbol. Ak taký nie je, tak sa všetko vráti do podoby pred volaním tento funkcie. Urobí príslušné zmeny počtov krížikov.
- `findNewSymbol()` - nájde a vráti voľný symbol. Ak neexistuje voľný symbol, tak vráti `None`
- `SymbolIsUsed(sym)` - vráti `true`, ak je symbol `sym` použitý. V opačnom prípade vráti `false`.

Kapitola 2

Výber palety

Na vyšitie jednej krížikovej výšivky sa zvyčajne používajú vyšívacie priadze jednej značky. Zaručí sa tým rovnaká kvalita, hrúbka a lesk na všetkých miestach výšivky. Preto je prirodzené, že sme používateľovi v našej aplikácii umožnili zvoliť si značku vyšívacej priadze, ktorú chce vo svojej výšivke použiť. Tým nám jednoznačne určí paletu. Táto paleta má však zvyčajne niekoľko stoviek vyšívacích priadzí. Nebolo by praktické, a ani sa to v praxi nepoužíva, keby sme vo výšivke používali všetky priadze palety. Preto z tejto veľkej palety musíme vybrať menšiu, ktorá má len maximálne niekoľko desiatok výšívacích priadzí. Farby vyšívacích priadzí v tejto palete by mali čo najlepšie pokrývať farby vo vstupnej bitmape.

Vstupom pre tento problém je bitmapa B rozmeru $m \times n$, paleta[1] P_{IN} a maximálny počet vyšívacích priadzí max, ktoré chceme vo výšivke použiť. Výstupom je paleta P_{OUT}, ktorá je podmnožinou palety P_{IN}, jej mohutnosť je najviac max a čo najviac pokrýva farby bitmapy B. Pokiaľ to hodnota max dovoľuje (max nie je príliš malé), tak by mala výstupná paleta spĺňať nasledujúce podmienky:

1. Ku každej výraznej a primerane často sa vyskytujúcej farbe by mala obsahovať vyšívaciu priadzu príbuznej farby.

2. Ak sa v bitmape nachádzajú plynulé prechody z jednej farby do druhej, tak by mala paleta obsahovať niekoľko vyšívacích priadzí, ktoré rovnomerne pokrývajú tento prechod.

Tieto dve podmienky by mali byť vyvážené. Ani jeden extrém nie je dobrý.

2.1 Prvé riešenie

Prvé riešenie vzniklo z myšlienky, že farby, ktoré sú v bitmape najčastejšie, by mali byť aj vo výšivke.

[1] zvyčajne všetky vyšívacie priedze jednej značky

Algoritmus pre prvé riešenie je rozdelený na dve časti.

1. Prvou časťou je štatistika. Pre každý pixel $(x, y)_B$ nájdeme $\Phi_{P_{IN}}(p_B(x, y))$ a spočítame koľko sme priradili ktorej priadzi. Presnejšie štatistika je zobrazenie $C : P_{IN} \to \mathbb{N}$ také, že

$$\forall t \in P_{IN} : C(t) = |\{(x, y)_B \mid \Phi_{P_{IN}}(p_B(x, y)) = t\}|$$

2. Počas druhej fázy redukujeme paletu na požadovanú veľkosť. Priadze budeme vyhadzovať po jednom. Konkrétne, nájdeme priadzu t_1, ku ktorej bolo farebne najbližšie najmenej pixelov. Túto priadzu z palety odstránime a pixely, ktoré pod ňu patrili, priradíme jej farebne najbližšej priadzi zo zvyšnej množiny. Presnejšie vytvárame postupnosť paliet $\{P_i\}_{i=0}^{\infty}$ a postupnosť zobrazení $\{C_i : P_i \to \mathbb{N}\}_{i=0}^{\infty}$. Pre P_0 a C_0 sú nasledovné:

$$P_0 = \{t \in P_{IN} \mid C(t) > 0\}$$
$$\forall t \in P_0 : C_0(t) = C(t)$$

   Ďalšie P_{i+1} a C_{i+1} pre $i = 0, 1, 2, \ldots$ vytvárame nasledovne: Nech $t_1 \in P_i$ je taká, že $\forall t \in P_i : C_i(t_1) \leq C_i(t)$. Potom

$$P_{i+1} = P_i \setminus t_1$$

   Nech $t_2 = \Phi_{P_{i+1}}(F(t_1))$. Potom

$$\forall t \in P_{i+1} \setminus t_2 : C_{i+1}(t) = C_i(t)$$
$$C_{i+1}(t_2) = C_i(t_2) + C_i(t_1)$$

   Výstupom je $P_{OUT} = P_N$ kde $N = maximum(|P_0| - max, 0)$.

Dôvodom, prečo sme pri odstraňovaní priadze z palety "prelievali"pixely najbližšej farbe, je, že takto aj menej početné farby dostali šancu sa dostať do palety. Jednoducho sa mohlo stať, že ak tam je veľa rôznych odtieňov nejakej farby, ale ani jeden z nich nie je veľmi početný, tak sa mohli spojiť do jednej farby, ktorá je už oveľa početnejšia a tým pádom ju tento algoritmus nevyhodí.

Bohužiaľ toto opatrenie pre niektoré obrázky nestačí a tak nemôžeme tvrdiť, že dobre spĺňa prvú podmienku popísanú v úvode tejto kapitoly. Často sa totiž stáva, že výraznej farby, ktorú treba mať v palete nie je veľa. Keďže tento algoritmus nepozerá na farby, ale len na počty, tak túto farbu jednoducho vyhodí. Avšak toto riešenie nie je až také zlé, lebo celkom dobre spĺňa druhú podmienku, ak je tieňovaná plocha dostatočne veľká.

2.2 Druhé riešenie

Riešenie popísané v tejto kapitole sa snaží aspoň trochu vylepšiť to predchádzajúce. Pri navrhovaní tohto riešenia sme použili fakt, že niektoré farby vyšívacích priadzí sú natoľko podobné, že sa takmer nedajú odlíšiť. Preto je zbytočné, aby boli vo výstupnej palete obe takéto vyšívacie priadze.

Algoritmus je tentokrát rozdelený až na tri časti:

1. Prvá časť je rovnaká ako v predchádzajúcom riešení. Pre každý pixel nájdeme farebne najbližšiu vyšívaciu priadzu. Zapamätáme si, koľko pixelov spadá pod ktorú priadzu. Použijeme rovnaké označenie ako v predchádzajúcej kapitole: $C(t)$ je počet pixelov, ktoré spadajú pod priadzu $t \in P$

2. Druhá časť je tiež rovnaká ako v predchádzajúcej kapitole. Postupne z palety odstraňujeme priadzu, pod ktorú spadá najmenej pixelov. Zaviedli sme si však koeficient $k \in [0, 1]$, ktorý určí hranicu, kedy priadzu ešte odstránime a kedy prejdeme do ďalšej časti algoritmu. Ak pod ňu spadá menej ako $100k\%$ pixelov bitmapy, tak ju odstránime. V opačnom prípade prejdeme do tretej časti. Presnejšie opäť vytvárame postupnosť paliet $\{P_i\}_{i=0}^{\infty}$ a postupnosť zobrazení $\{C_i : P_i \to \mathbb{N}\}_{i=0}^{\infty}$, pričom C_0 aj P_0 sú rovnaké ako v predchádzajúcom riešení. Ďalšie P_{i+1} a C_{i+1} pre $i = 0, 1, 2, \ldots$ vytvárame nasledovne:
   Nech $t_1 \in P_i$ je taká priadza, že $\forall t \in P_i : C_i(t_1) \leq C_i(t)$. Ak $C_i(t_1) \geq m \times n \times k$, tak prejdi na ďalšiu časť algoritmu. Inak:

$$P_{i+1} = P_i \setminus t_1$$

   Nech $t_2 = \Phi_{P_{i+1}}(F(t_1))$. Potom

$$\forall t \in P_{i+1} \setminus t_2 : C_{i+1}(t) = C_i(t)$$
$$C_{i+1}(t_2) = C_i(t_2) + C_i(t_1)$$

3. V poslednej časti budeme opäť redukovať počet pridzí v palete po jednej. Tentokrát ale budeme spájať priedze, ktoré sú farebne najbližšie. V palete potom zostane priadza, pod ktorú spadá viac pixelov. Budeme pokračovať v budovaní postupností, ktoré sme začali vyrábať v druhej časti algoritmu.
   Nech $t_1, t_2 \in P_i$ su také, že $\forall t, u \in P_i : d(F(t_1), F(t_2)) \leq d(F(t), F(u))$.
   Ak $C_i(t_1) < C_i(t_2)$, tak

$$P_{i+1} = P_i \setminus t_1$$
$$\forall t \in P_{i+1} \setminus t_2 : C_{i+1}(t) = C_i(t)$$
$$C_{i+1}(t_2) = C_i(t_2) + C_i(t_1)$$

V opačnom prípade

$$P_{i+1} = P_i \setminus t_2$$
$$\forall t \in P_{i+1} \setminus t_1 : C_{i+1}(t) = C_i(t)$$
$$C_{i+1}(t_1) = C_i(t_1) + C_i(t_2)$$

Výstupom je $P_{OUT} = P_N$, kde $N = |P_0| - max$.

Ak $k = 0.01$, tak toto riešenie dávalo lepšie výsledky ako prvé riešenie. Výsledky sme porovnávali len vizuálne. Ak za k zvolíme 1, tak dostaneme prvé riešenie, lebo podmienka prechodu na tretiu časť algoritmu nebude nikdy splnená. Ak zvolíme $k = 0$, tak sa z palety odstránia takmer všetky odtiene a zostanú len pridze, ktoré sú farebne ďaleko od seba. Ani jedeno z týchto nastavení nemusí byť zlé pre rôzne typy obrázkov.

Pri testovaniach sme zistili, že pre fotografie pri nastavení $k = 0.01$, sa v druhej fáze algoritmu odstráni z palety až okolo 90% priadzí z tých, ktoré treba odstrániť. Z toho je veľa takých, ku ktorým prislúchal len jeden pixel bitmapy. To je aj dôvod, prečo aj toto riešenie spĺňa primerane druhú podmienku z úvodu tejto kapitoly. Prvú podmienku spĺňa lepšie ako predchádzajúce riešenie (pre nastavenie $k = 0.01$), aj keď určite existujú obrázky v ktorých je výrazná farba menej ako jedno percento a preto ju aj tento algoritnmus z palety odstráni.

2.3 Tretie riešenie

Doteraz sme vybraté palety porovnávali iba vizuálne. Potrebujeme však nejaký ukazovateľ, ktorým by sme vedeli jednoducho porovnať vygenerované palety. Celkom dobrým ukazovateľom je súčet druhých mocnín vzdialeností medzi farbou pixela a najbližšou farbou vyšívacej priadze z palety. Tento koeficient by mal byť nižší pre palety, ktoré spĺňajú podmienky z úvodu kapitoly.

Definícia. Nech m, n sú rozmery vstupnej bitmapy B. Potom *koeficient podobnosti* pre paletu P k bitmape B je

$$\kappa_P = \sum_{x=1}^{m} \sum_{y=1}^{n} d^2(p_B(x, y), F(\Phi_P(p_B(x, y)) \tag{2.1}$$

Neprirodzené na predchádzajúcich riešeniach bolo to, že keď sme odstraňovali priadzu z palety, pixely, ktoré jej prislúchali, sme automaticky všetky dali jej farebne najbližšej priadzi z ostatných. Prirodzenejšie by bolo, keby sme pixely opäť prerozdelili medzi ostatné farby. Takto bude stále platiť, že vždy pod priadzu bude patriť toľko pixelov, koľko jej naozaj prislúcha, keby sme urobili štatistiku znovu.

Toto sme zakomponovali do nášho tretieho riešenia, ktoré je len miernou modifikáciou druhého. Pri odstraňovaní priadze z palety budeme prerozdeľovať jej pixely medzi ostatné priadze. Keď sa budeme rozhodovať, ktorú priadzu z dvoch máme vyhodiť, vyskúšame obe možnosti a zoberieme tú, ktorá je lepšia vzhľadom na koeficient podobnosti. Riešenie je rozdelené na tri časti.

1. Prvou časťou je opäť štatistika. Je však trochu iná ako štatistika v predchádzajúcich riešeniach. Tentokrát si budeme namiesto počtov pamätať množiny pixelov, ktoré pod jednotlivé priadze spadajú. Štatistika je teda zobrazenie $X : P_{IN} \to 2^{\{1,2,\ldots,m\}\times\{1,2,\ldots,n\}}$ také, že:

   $$\forall t \in P_{IN} : X(t) = \{(x,y)_B | \Phi_{P_{IN}}(p_B(x,y)) = t\}$$

   Platí, že $\kappa_{P_{IN}} = \sum_{t\in P_{IN}} \sum_{(x,y)_B \in X(t)} d^2(F(t), p_B(x,y))$.

   V implementácii si okrem množiny pixelov pamätáme aj to, koľko prispievajú do koeficientu podobnosti. Pri odstraňovaní jednej farby potom netreba hľadať najbližšiu farbu pre všetky pixely, ale len pre tie, ktoré premiestňujeme. Z koeficientu podobnosti sa potom odstráni podiel odstraňovanej farby a pridajú sa mocniny vzdialeností medzi farbami prerozdeľovaných pixelov a k nim najbližšími farbami vyšívacích priadzí, ktoré zostávajú v palete.

2. V druhej časti budeme opäť z palety postupne odstraňovať priadze, pod ktoré spadá najmenej pixelov. Pixely priadze, ktorú odstraňujeme však nedáme farebne najbližšej priadzi, ale ich rozdelíme medzi zvyšné priadze. Budeme mať koeficient k, ktorý bude mať rovnakú funkciu ako v druhom riešení. Pri odstraňovaní budeme robiť dve postupnosti. Budú to postupnosť paliet $\{P_i\}_{i=0}^{\infty}$ a postupnosť zobrazení $\{X_i : P_i \to 2^{\{1,2,\ldots,m\}\times\{1,2,\ldots,n\}}\}_{i=0}^{\infty}$ pričom $P_0 = \{t \in P_{IN} \mid |X(t)| > 0\}$ a $\forall t \in P_0 : X_0(t) = X(t)$. Ďalšie P_{i+1} a X_{i+1} pre $i = 0, 1, 2, \ldots$ vypočítame nasledovne:
   Nech $t_1 \in P_i$ je taká, že $\forall t \in P_i : |X_i(t_1)| \leq |X_i(t)|$. Ak $|X_i(t_1)| \geq m \times n \times k$, tak prejdi na tretiu časť riešenia. Inak

   $$P_{i+1} = P_i \setminus t_1$$
   $$\forall t \in P_{i+1} : X_{i+1}(t) = X_i(t) \cup \{(x,y)_B \in X_i(t_1) \mid \Phi_{P_{i+1}}(p_B(x,y)) = t\}$$

3. Treťou časťou algoritmu je hľadanie farebne najbližších priadzí a odstránenie jednej z nich. Pri rozhodovaní vyskúšame obe možnosti a zvolíme tú, pre ktorú bude menší koneficient podobnosti. Ak sa budú koeficienty zhodovať, tak rozhodne počet pixelov. Vtedy odstránime priadzu, pod ktorú patrí menej pixelov. Presnejšie ďalej budujeme postupnosti, ktoré sme začali budovať v predchádzajúcej časti.

Nech $t_1, t_2 \in P_i$ sú také priadze, pre ktoré platí $\forall t, u \in P_i : d(F(t_1), F(t_2)) \leq d(F(t), F(u))$. Ďalej pokračujeme vyskúšaním oboch variant. Nech

$$\begin{aligned} P^1 &= P_i \setminus t_1 \\ \forall t \in P^1 : X^1(t) &= X_i(t) \cup \{(x,y)_B \in X_i(t_1) \mid \Phi_{P^1}(p_B(x,y)) = t\} \\ P^2 &= P_i \setminus t_2 \\ \forall t \in P^2 : X^2(t) &= X_i(t) \cup \{(x,y)_B \in X_i(t_2) \mid \Phi_{P^2}(p_B(x,y)) = t\}. \end{aligned}$$

Potom

$$\begin{aligned} \kappa_{P^1} &= \sum_{t \in P^1} \sum_{(x,y)_B \in X^1(t)} d^2(F(t), p_B(x,y)) \\ \kappa_{P^2} &= \sum_{t \in P^2} \sum_{(x,y)_B \in X^2(t)} d^2(F(t), p_B(x,y)) \end{aligned}$$

Ak $\kappa_{P^1} < \kappa_{P^2}$ alebo $\kappa_{P^1} = \kappa_{P^2} \wedge |X_i(t_1)| > |X_i(t_2)|$, tak

$$\begin{aligned} P_{i+1} &= P^2 \\ X_{i+1} &= X^2 \end{aligned}$$

V opačnom prípade

$$\begin{aligned} P_{i+1} &= P^1 \\ X_{i+1} &= X^1 \end{aligned}$$

Výstupom je $P_{OUT} = P_N$ kde $N = maximum(|P_0| - max, 0)$.

Ak zvolíme za $k = 0.01$, tak toto riešenie spĺňa kritériá z úvodu kapitoly trochu lepšie ako predchádzajúce riešenia. Prvé kritérium spĺňa podobne dobre ako predchádzajúce riešenie. Počty vyhodených farieb v druhej fáze sa zhruba zhodujú. Zlepšenie sme však pri testoch spozorovali pri druhom kritériu. Toto riešenie totiž vyberá rovnomernejšie farby pre prechody.

Výstupy algoritmov pre tu spomenuté riešenia môžete vidieť na obrázkoch A.2, A.3 a A.4. Ku každej palete tam nájdete koeficient podobnosti, počet priadzí odstránených v druhej fáze a v tretej fáze a obrázok upravený do farieb priadzí palety.

Kapitola 3

Zmena veľkosti vstupnej bitmapy

V tejto kapitole sa budeme venovať problému zmeny rozmerov vstupnej bitmapy na rozmery výšivky tak, aby jeden pixel výstupnej bitmapy zodpovedal jednému krížiku výšivky. Tento problém potrebujeme vyriešiť, lebo algoritmy, ktoré budeme aplikovať neskôr, už rozmery bitmapy nemenia a väčšina z nich sa nedala jednoducho upraviť tak, aby ich menila.

Vstupnú bitmapu budeme označovať B_{IN} a výstupnú B_{OUT}. Predpokladáme, že pomer dĺžok strán vstupnej bitmapy je rovnaký ako pomer dĺžok strán výstupnej bitmapy. Keby to tak nebolo, tak by mohol byť obraz vo výšivke deformovaný, keďže jednému pixelu výstupnej bitmapy bude zodpovedať jeden krížik výšivky. Aby sa nestalo, že používateľ zadá rozmery v zlom pomere, dali sme mu možnosť nastaviť iba jeden ľubovoľný rozmer. Druhý rozmer sa dopočíta.

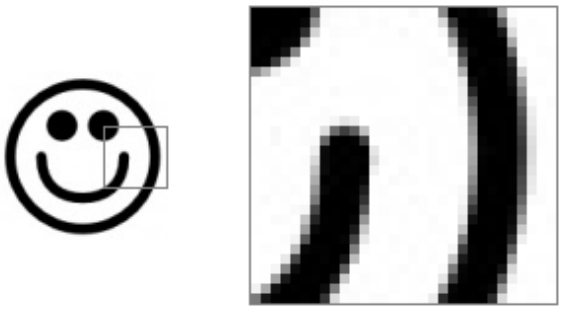
Bez ujmy na všeobecnosti, nech je zadaná šírka výšivky v krížikoch $W_{B_{OUT}}$. Rozmery bitmapy B_{IN} v pixeloch sú $W_{B_{IN}} \times H_{B_{IN}}$. Potom dĺžka strany štvorčeka, ktorý predstavuje jeden pixel výstupnej bitmapy B_{OUT} v pixeloch je $\delta = W_{B_{IN}} / W_{B_{OUT}}$. Výšku výstupnej bitmapy $H_{B_{OUT}}$ vypočítame ako $H_{B_{OUT}} = \lfloor H_{B_{IN}} / \delta \rfloor$. Tým pádom stačí zadať dĺžku δ strany štvorčeka v pixeloch a rozmery výšivky, a teda aj výstupnej bitmapy, sa dopočítajú.

Riešenia tohto problému by mali spĺňať tieto kritériá:
Výstupná bitmapa by mala čo naviac zachovať

1. tvary a farebnosť vstupnej bitmapy.
2. plynulé farebné prechody vstupnej bitmapy
3. farebné skoky vstupnej bitmapy

Treba aj myslieť na to, že farebné skoky, ktoré sa nám zdajú byť jasné v bitmape s malými pixelmi sa nemusia zdať také jasné aj v tej istej zväčšenej bitmape. Často sa tam nachádzajú prechody na jednom alebo dvoch radoch pixelov, ktoré pri malých pixeloch nie je vidno. Môžete to vidieť na obrázku 3.1. Jednoducho sa využíva schopnosť ľudského

oka miešať farby. Na túto schopnosť sa pri výšivkách nemôžeme úplne spoliehať. Krížiky, ktoré sú vo výšivke vlastne naše pixely, sú príliš veľké. Jeden krížik má približne 2×2 *mm*. Vyzeralo by nepekne, keby boli vo výšivke hranice riešené tak, ako v bežných bitmapách. Hranice jednoducho musia byť konkrétne. Tou treťou podmienkou sme mysleli práve to.



Obr. 3.1: Ukážka zväčšenej bitmapy

3.1 Priemerovanie

Prvým nápadom na riešenie tohto problému bolo jednoduché priemerovanie farieb pixelov, ktoré pokryje jeden pixel výstupnej bitmapy.

Pre každý pixel výstupnej bitmapy $(x, y)_{B_{OUT}}$ najprv musíme vypočítať hranice obdĺžnikovej oblasti (nemusí byť vždy štvorcová), z ktorej tento pixel vznikne. Budeme pracovať so sprava otvorenými intervalmi. Hranice si označíme x_0, x_1, y_0, y_1, pričom pixely, ktoré budeme neskôr priemerovať budú mať súradnice z množiny $\{x_0, x_0 + 1, \ldots, x_1 - 1\} \times \{y_0, y_0 + 1, \ldots, y_1 - 1\}$. Pri výpočte týchto hraníc použijeme dĺžku strany štvorca δ vypočítanú vyššie.

$$\begin{aligned} x_0 &= \lfloor x\delta \rfloor \\ x_1 &= \lfloor (x+1)\delta \rfloor \\ y_0 &= \lfloor y\delta \rfloor \\ y_1 &= \lfloor (y+1)\delta \rfloor \end{aligned}$$

Ak $x_0 = x_1$, tak x_1 zväčšíme o jedna, aby oblasť prislúchajúca ku pixelu $(x, y)_{B_{OUT}}$ nebola prázdna. Podobne, ak $y_0 = y_1$, tak y_1 zväčšíme o jedna.

Teraz spočítame priemernú farbu na vypočítanej oblasti. Keďže používame farebný model RGB, môžeme priemernú farbu počítať po zložkách, pričom farbu chápeme ako trojrozmerný vektor.

$$p_{B_{OUT}}(x, y) = \left\lfloor \frac{\sum_{i=x_0}^{x_1-1} \sum_{j=y_0}^{y_1-1} p_{B_{IN}}(i, j)}{(x_1 - x_0)(y_1 - y_0)} \right\rfloor \tag{3.1}$$

pričom dolnú celú časť aplikujeme po zložkách. V prípade, že by sme chceli použiť iný farebný model, bolo by treba prehodnotiť, či je takýto spôsob priemerovania farieb vhodný.

Toto riešenie dobre spĺňa prvé dve kritériá. Avšak s tretím kritériom má problémy hlavne pre $\delta > 1$. Vtedy do oblasti pre jeden pixel výstupnej bitmapy spadá viac ako jeden pixel vstupnej bitmapy. Ak takouto oblasťou prechádza hrana, tak tento algoritmus zmieša farby aj na jednej, aj na druhej strane hrany, čím ju rozmaže. Ak cez túto oblasť dokonca prechádza nejaká čiara, tak ju v lepšom prípade stlmí a v horšom úplne odstráni. V zásade toto riešenie nie je zlé a pre niektoré vstupné bitmapy úplne postačuje. Napríklad pre také, ktoré neobsahujú farebné skoky.

3.2 Detekcia hrán

Priemerovanie v princípe nie je zlý postup, ale bolo by ho treba nejako upraviť tak, aby nerozmazával hrany. Ak to chceme urobiť, musíme pri priemerovaní jednu stranu hrany uprednostniť.

Aby sme mohli jednu stranu hrany uprednostnit, musíme najprv hrany nájsť. Tento problém sa po anglicky nazýva edge detection. Existuje mnoho metód, ktoré hľadajú hrany s rôznymi výsledkami. My však potrebujem taký spôsob, ktorý nájde hranu a uprednostní iba jednu jej stranu. Takúto vlastnosť má takzvaný operátor Laplacian.

Podľa [3] je Laplacian pre funkciu $f(x, y)$ dvoch premenných definovaný ako

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{3.2}$$

V našom prípade platí $f(x, y) = p_{B_{IN}}(x, y)$. Aby bol tento operátor užitočný pre náš problém, musíme túto rovnicu vyjadriť v diskrétnej forme. V [3] to vyjadrili nasledovne:

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \tag{3.3}$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \tag{3.4}$$

$$\nabla^2 f = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \tag{3.5}$$

Toto vieme implementovať pomocou masky na obrázku 3.2, pričom spracúvaný pixel je v jej strede.

0	1	0
1	-4	1
0	1	0

Obr. 3.2: Diskrétna forma operátora Laplacian

Takýto filter zachytáva hrany len v dvoch smeroch a to vo vertikálnom a horizontálnom. Dajú sa pridať ešte ďalšie dva diagonálne smery [3]. V takom prípade by maska pre tento

operátor vyzerala ako na obrázku 3.3a. Okrem tohto tvaru sa v praxi používa aj tvar, ktorý môžete vidieť na obrázku 3.3b. Keď sa jedena z foriem operátora Laplacian aplikuje na bitmapu, tak hrany zostanú svetlé a zvyšok bitmapy je tmavý. Príkladom takto upravenej bitmapy je A.5b resp. A.5a. Na týchto obrázkoch si tiež môžete všimnúť rozdiel medzi vyššie spomenutými filtrami. Prvý filter (kladný) zvýrazňuje tmavú stranu hrany a druhý (záporný) naopak svetlú stranu hrany.

1	1	1
1	-8	1
1	1	1

(a) kladný

-1	-1	-1
-1	8	-1
-1	-1	-1

(b) záporný

Obr. 3.3: Diskrétne formy operátora Laplacian doplnené o diagonály

Teraz potrebujeme pre každý pixel vypočítať jeho váhu pri priemerovaní. Čím je pixel svetlejší po úprave operátorom Laplacian, tým väčšiu váhu by mal mať. Označme si bitmapu upravenú laplacian filtrom B_F. Táto bitmapa je farebná, ale potrebujeme z neho iba svetlosti pixelov, preto každému pixelu priradíme číslo (odtieň šedej) od 0 po 255

Je niekoľko možností, ako vypočítať odtieň šedej. My sme si zvolili priemerovanie farebných zložiek.

Definícia. Nech je daná farba $C = (r, g, b)$. Potom jej odtieň šedej je $g(C) = (r + g + b)/3$

Teraz by sme mohli jednoducho zobrať odtieň šedej a povedať, že to je váha daného pixela. Chceli sme však nechať používateľovi voľnosť v tom, ktorú stranu hrany uprednostní a ako veľmi. Preto sme ešte zaviedli parametrickú funkciu, ktorá podľa toho, ako veľmi chce používateľ jednu stranu hrany uprednostniť, priradí pixelom vstupej bitmapy váhy. Určili sme, že bude 6 stupňov uprednostnenia jednej strany hrany. Tieto stupne sú očíslované od 0 po 5. Čím väčšie číslo, tým viac sa uprednostní strana hrany, ktorú si užívateľ vyberie.

Nech k je stupeň uprednostnenia. Funkcia, ktorá určuje váhu pixela je lineárne funkcia $h : \{0, 1, \ldots, 255\} \to \mathbb{R}$ taká, že $h(255) = 2^5$ a $h(0) = 2^{5-k}$. Z toho vyplýva, že h má nasledovný predpis.

$$h(x) = \left(\frac{2^5 - 2^{5-k}}{255}\right)x + 2^{5-k}$$

Teraz zhrnieme náš algoritmus.

1. Pre každý pixel $(x, y)_{B_{OUT}}$ vypočítame oblasť, ktorú bude pokrývať. Túto oblasť vypočítame rovnako ako v predchádzajúcom riešení. Okraje tejto oblasti opäť označíme x_0, x_1, y_0, y_1.

2. Pre každý $(x, y)_{B_{IN}} \in \{x_0, x_0 + 1, \ldots, x_1 - 1\} \times \{y_0, y_0 + 1, \ldots, y_1 - 1\}$ vypočítame farbu s použitím operátora Laplacian. Podľa toho, čo chce používateľ uprednostniť, použime príslušnú masku. Nech chce uprednostniť tmavú stranu hrany. Použijeme teda masku z obrázku 3.3a:

$$p_{B_F}(x, y) = p_{B_{IN}}(x - 1, y - 1) + p_{B_{IN}}(x, y - 1) + p_{B_{IN}}(x + 1, y - 1) + p_{B_{IN}}(x - 1, y) + p_{B_{IN}}(x + 1, y) + p_{B_{IN}}(x - 1, y + 1) + p_{B_{IN}}(x, y + 1) + p_{B_{IN}}(x + 1, y + 1) - 8p_{B_{IN}}(x, y)$$

   V opačnom prípade použijeme ten istý vzorec len prenásobený mínus jednotkou. Pre okrajové pixely si tento vzorec upravíme tak, že koeficient pri $p_{B_{IN}}(x, y)$ je minus počet jeho susedov (maximum je 8) pre kladný Laplacian, a plus počet susedov pre záporný Laplacian.

   Potom z tohto pixela vygenerujeme jeho odtieň šedej a k nemu dopočítame jeho váhu. Nech stupeň uprednostnenia je k. Váhou tohto pixela je $v(x, y) = h(g(p_{B_F}(x, y)))$, pričom h a g sú funkcie spomenuté vyššie.

3. Nakoniec vypočítame farbu pixela $(x, y)_{B_{OUT}}$ podľa nasledujúceho vzorca:

$$p_{B_{OUT}}(x, y) = \left\lfloor \frac{\sum_{i=x_0}^{x_1-1} \sum_{j=y_0}^{y_1-1} v(i, j) p_{B_{IN}}(i, j)}{\sum_{i=x_0}^{x_1-1} \sum_{j=y_0}^{y_1-1} v(i, j)} \right\rfloor$$

   Tento vzorec vlastne opisuje váhované priemerovanie farieb pixelov s váhami vypočítanými vyššie.. Samozrejme dlnú celú časť zase aplikujeme na jednotlivé farebné zložky zvlášť.

Takto upravené priemerovanie je oveľa univerzálnejšie ako to predchádzajúce. Ak za koeficient k zvolíme 0, tak je to obyčajné priemerovanie, lebo funkcia h je konštanta. Čo sa týka kritérii, druhé je splnené rovnako ako pri priemerovaní. V tieňovanej ploche totiž nie sú hrany. Preto je celá táto plocha po úprave Laplacianom čierna a všetky jej pixely majú rovnakú váhu. Na takéto plochy teda stále používame obyčajné priemerovanie. Tretie kritérium je splnené podľa toho, aký je zvolený koeficient k. Čím je vyšší, tým viac sú hrany konkrétne. Čo sa tým trochu poškodzuje je prvé kritérium. Farebnosť je v poriadku, ale tvary sa môžu deformovať. Keď chceme napríklad veľmi uprednostniť tmavú stranu hrany, všetky tmavé ohraničené plochy budú zrazu väčšie. Napríklad aj tenká čiara sa môže zmeniť na druhú. Podobne je to pri uprednostňovaní svetlejšej časti hrany. Preto nechávame na uvážení používateľa ako veľmi a čo uprednostní, aby dosiahol želaný výsledok.

Ukážky výstupov tohto algoritmu môžete vidieť na obrázkoch A.5, A.6 a A.7.

Kapitola 4

Priradenie priadzí krížikom výšivky

Posledným problémom, ktorý je treba vyriešiť je priradenie vyšívacích priadzí jednotlivým krížikom. Inak povedané, určíme pre každý krížik, akou farbou má byť vyšitý. To znamená, že z vybratej palety P a bitmapy B v rozmeroch výšivky $m \times n$ vyrobíme predlohu $R = (M, S)$, pričom matica M má rovnaké rozmery ako bitmapa B. Hlavným problémom pri generovaní predlohy je v tomto prípade naplnenie matice M. Zobrazením S sa nebudeme zaoberať. Ako ste si mohli všminúť v implementácii triedy `Pattern`, symboly pre vyšívacie priadze sa priraďujú automaticky počas nastavovania prvkov matice. Navyše používateľ má možnosť tieto symboly meniť.

Priradenie priadzí krížikom by malo spĺňať nasledujúce podmienky:

1. Zachovať farebnosť ako najviac vybratá paleta dovoľuje
2. Plynulé farebné prechody by mali byť aj naďalej pôsobiť plynulo
3. Farebné skoky, by mali zostať skokmi.

Celkovo, by sa mala výšivka čo najviac podobať na vstupnú bitmapu.

4.1 Nahradenie najbližšou farbou

Najjednoduchší spôsob ako naplniť maticu je nájsť pre každý pixel bitmapy B jeho farebne najbližšiu priadzu zo vstupnej palety P a túto priadzu potom uložiť do matice M na príslušné miesto. Presnejšie:

$$\forall (x, y) \in \{1, 2, \ldots, m\} \times \{1, 2, \ldots, n\} : M(x, y) \quad = \quad \Phi_P(p_B(x, y))$$

Takéto riešenie sa teda snaží čo najviac zachovať farebnosť. Preto veľmi dobre spĺňa prvé kritérium. Taktiež veľmi dobre spĺňa aj tretie kritériu, pokiaľ paleta poskytuje vyšívacie priadze vhodných farieb. Takže toto riešenie je výborné pre bitmapy, ktoré neobsahujú

plynulé farebné prechody. Ak tam také farebné prechody sú, tak sa na týchto prechodoch vytvoria veľké plochy jednej farby. Navyše tieto plochy majú veľmi často veľmi konkrétne hranice. Ako môžete vidieť na obrázku A.9a. Preto tento prvý algoritmus nespĺňa druhé kritérium. Skúmali sme teda ďalšie možnosti riešenia tohto problému, ktoré by spĺňali aj druhé kritérium.

4.2 Dithering

Riešenie, ktoré sme našli je použitie tzv. ditheringu. V slovenčine sa mu tiež hovorí rozptyl. Dithering je pojem, za ktorým sa krývajú rôzne postupy. Všetky však majú rovnaký cieľ. Dithering slúži na zachovanie kvality obrázku, aj keď je k dispozícií málo farieb. Kvalita je v zmysle ľudkého vnímania. Špeciálnym prípadom je tzv. digital halftoning. Tento digital halftoning je spôsob, ako vyrobiť zo šedotónového obrázku čisto čiernobiely. Pán Ulichney [10] uviedol jeho definíciu: Digital Halftoning je "každý algoritmický proces, ktorý vytvára ilúziu obrazu so spojitými tónmi z rozumného usporiadania binárnych prvkov obrazu."Dithering je vlastne to isté, len vo farebnejšom prevedení. Takéto postupy sú dôležité hlavne pre zariadenia, ktoré majú obmedzené možnosti zobrazovania farieb. V každom prípade sa však využíva schopnosť ľudského zraku miešať farby, keď je veľa malých kúskov rôznych farieb na jednom priestore.

Toto by sme chceli využiť aj pre náš problém. Použitím ditheringu by sme mohli vyriešiť problém hlavne s veľkými ucelenými plochami namiesto plynulého prechodu, ktorý malo predchádzajúce riešenie.

Teraz nám stačí vybrať ten správny typ ditheringu pre náš problém. Asi najznámejšie typy ditheringu sú random dithering (náhodný rozptyl), ordered dithering (maticový rozptyl), error diffusion dithering (rozptyl s distribúciou odchýlky) a Riemersma dithering.

Random dithering je z týchto všetkých typov ditheringov najmenej kvalitný. Ako jeho názov napovedá, veľa používa náhodu a preto obrázok, upravený týmto ditheringom, obsahuje veľa šumu. To nie je vlastnosť, ktorú potrebujeme, preto sme sa tento dithering ani nepokúsili použiť.

Ordered dithering bohužiaľ nemžeme použiť. Tento dithering totiž predpokladá, že farby palety sú rovnomerne rozmiestnené vo farebnom priestore a my túto vlastnosť nemôžeme zaručiť. Keby sa však dal použiť, mohol by byť pre vyšívanie veľmi zaujímavý, lebo vytvára celkom pekné pravidelné vzory.

Ditheringy, ktoré môžeme použiť sú error diffusion dithering a Riemersma dithering. Tieto ditheringy totiž nič nepredpokladajú o palete, ktorú dostanú. Každému z týchto ditheringov teraz venujeme jednu podkapitolu.

4.2.1 Error diffusion dithering

Error diffusion dithering je dithering, ktorý na miešanie farieb používa prenášanie chyby, ktorá vznikne priradením novej farby pixelu, na susedné pixely podľa daného filtra. Je veľmi veľa filtrov, ktoré sa dajú použiť. Všetky, ktoré sme v našej práci použili, boli pôvodne navrhnuté pre digital halftoning. Môžete ich vidieť na obrázku 4.1. Znakom $*$ sa vždy označuje aktuálne spracúvaný pixel. Čísla oko neho hovoria, koľko chyby treba danému susedovi predať. Všetky tieto čísla sú samozrejme ešte prenásobené koeficientom v zátvorke.

Aj keď boli tieto filtre pôvodne navrhnuté pre digital halftoning, tak sa dajú použiť aj pre farebné obrázky. Stačí použiť RGB model a filter aplikovať na každú zložku farby zvlášť. Existujú články, ktoré skúmajú práve to, do akej miery sú tieto filtre vhodné pre farebné obrázky. My sme sa skôr zaoberali tým, ktorý z týchto filtrov je vhodný pre náš problém.

		*	7	
3	5	1		(1/16)

(a) Floyd-Steinberg [2]

			*	7	5
3	5	7	5	3	
1	3	5	3	1	(1/48)

(b) Jarvis, Judice, Ninke [5]

			*	8	4
2	4	8	4	2	
1	2	4	2	1	(1/42)

(c) Stucki [9]

			*	8	4
2	4	8	4	2	
					(1/32)

(d) Burkes[1]

		*	1	1
1	1	1		
		1		(1/8)

(e) Atkinson

			*	5	3
2	4	5	4	2	
		2	3	2	(1/32)

(f) Sierra [8]

			*	4	3
1	2	3	2	1	(1/16)

(g) Two-Row Sierra [8]

	*	2	
1	1		(1/4)

(h) Sierra Light [8]

Obr. 4.1: Rôzne filtre pre error diffusion dithering

Teraz presnejšie popíšeme algoritmus aplikovania filtra na vstupnú bitmapu. Ilustrujme to na filtri Floyd-Steinberg (obrázok 4.1a). Algoritmus tentokrát uvedieme pomocou pseudokódu. Vstupnú paletu P budeme používať ako inštanciu triedy `ColorPalette` z prvej

kapitoly. Vstupnú bitmapu *B* budeme používať ako dvojrozmerné pole farieb. Farby sú pre nás trojrozmerné vektory, takže všetky operácie s nimi, robíme po zložkách. Výstupná predloha *R* je inštancia triedy `Pattern` z prvej kapitoly.

```
1  pre kazdy riadok y:
2     pre kazdy stplec x:
3        id = P.getNearestIDToColor(B[y][x])
4        R.setSquare(x,y,id)
5        newcolor = P.getColor(id)
6        error = B[y][x] - newcolor
7        B[y][x+1]   = B[y][x+1]   + error*7/16
8        B[y+1][x-1] = B[y+1][x-1] + error*3/16
9        B[y+1][x]   = B[y+1][x]   + error*5/16
10       B[y+1][x+1] = B[y+1][x+1] + error*1/16
```

V tomto pseudokóde ešte chýbajú overenia, či susedný pixel naozaj existuje, ale pre prehľadnosť kódu sme ich neuviedli.

Od filtra vhodného pre náš problém očakávame, že nebude miešať veľmi kontrastné farby, pokiaľ to vyslovene nie je nutné. Jednoducho vo výšivke nevyzerá dobre, keď sú v nejakej farbe bodky inej kontrastnej farby. Pri malých pixeloch to možno nevadí, ale my nesmieme zabúdať na veľkosť našich skutočných pixelov (krížikov).

Pri testovaní sme zistili, že najlepšie túto podmienkú spĺňa Atkinson filter. Tento filter je špeciálny v tom, že neprenáša celú chybu, ale len jej 3/4. Navyše túto chybu rozdelí až na šesť susedov, čím spôsobí, že sa chyba až tak veľmi nekumuluje. V prípade konverzie šedotónového obrazu na čiernobiely, nemá tendenciu miešať veľa bielych pixelov do čiernej, ak je pôvodná oblasť dosť tmavá a naopak. V prípade farebných obrázkov, ak sa v palete nachádza k nejakej farbe na veľkej ploche dostatočne blízka farba, tak ju tam dá a nemieša ju s inými. To je výborná vlastnosť, lebo nech už budeme vyberať paletu akokoľvek dobre, vždy farby priadzí nebudú úplne zodpovedať farbám bitmapy.

V prípade ostatných ditheringov sa oveľa častejšie stávalo, že miešali kontrastné farby, aj keď to z pohľadu užívateľa nebolo treba. Stačila napríklad veľká plocha tmavosivej farby, ktorá bola trochu do zelena. K takejto farbe akurát nebola identická farba vyšívacej priadze. Farbene najbližšia k nej bola čistá sivá približne rovnakej svetlosti. Pri spracovaní bitmapy filtrami prenášajúcimi celú chybu sa na niektorých miestach nahromadila chyba tak veľmi, že sa namiesto sivej farby použila svietivá zelená, ktorá bola zhodou okolností tiež v palete. Vznikla tak veľká plocha šedej farby so svetlozelenými bodkami.

Porovnanie filtrov Floyd-Steinberg, Sierra Two-Row a Atkinson môžete vidieť na obrázkoch A.8, A.9 a A.10.

4.2.2 Riemersma dithering

Riemersma dithering sa snaží spojiť výhody error diffusion a ordered ditheringu. Konkrétne možnosť použiť akúkoľvek paletu pri error diffusion ditheringu a nezávislosť spracovania pixelov pri ordered ditheringu [7]. Nezávislosť spracovania pixelov od ostatných buhožiaľ nespĺňa ale robí aspoň kompromis.

Tento dithering je veľmi podobný error diffusion ditheringu v tom, že prenáša chybu. Túto chybu však neprenáša na niekoľko susedov, ale iba na jedného suseda. Tento sused sa vyberá podľa tzv. priestor vypĺňajúcej krivky. Najčastejšie sa používa Hilbertova krivka.

Hilbertova krivka

Asi najznámejšou priestor vypĺňajúcou krivkov je Hilbertova krivka. Pán Hilbert túto krivku definoval pomocou opakovaného špeciálneho číslovania štvorcov a ich následného rozdelenia každého švorca na štvrtiny [4]. Táto definícia je trochu volná. Existujú vždy až štyri možnosti ako môže vyzerať ďalšia iterácia. Štandardne sa však používa jeden tvar Hilbertovaj krivky, ktorá tvorí fraktal. Hilbertova krivka bola pôvodne navrhnutá na vyplnenie celého priestoru. Preto sa vždy pri ďalšej iterácii jeden krok zmenšoval na polovicu. Náš najmenší krok je z jedného pixela na jeho susedný. Preto krok nezmenšujeme, ale naopak zväčšujeme plochu.

Keďže je pre nás dôležitá postupnosť smerov, kam sa máme pohnúť, aby sme sa posunuli ďalej po krivke, definujeme ju rekurzívne. Výstupom bude postupnosť smerov.

$$
\begin{aligned}
D(n) &= R(n-1) \downarrow D(n-1) \rightarrow D(n-1) \uparrow L(n-1) \\
R(n) &= D(n-1) \rightarrow R(n-1) \downarrow R(n-1) \leftarrow U(n-1) \\
U(n) &= L(n-1) \uparrow U(n-1) \leftarrow U(n-1) \downarrow R(n-1) \\
L(n) &= U(n-1) \leftarrow L(n-1) \uparrow L(n-1) \rightarrow D(n-1) \\
D(0) &= R(0) = U(0) = L(0) = []
\end{aligned}
$$

pričom šípky určujú smer pohybu a zápis [] znamená prázdnu postupnosť. Pravé strany rovníc vždy reťazíme. Generovanie začíname volaním $D(n)$. Prvých päť iterácii pre $n = 1, 2, 3, 4, 5$ môžete vidieť na obrázku 4.2.

Môžete si všimnúť, že Hilbertova krivka vždy vyplní celý štvorec so stranou dlhou 2^n. Preto ak budeme chcieť pokryť všetky pixely vstupnej bitmapy, tak musíme nájsť také najmenšie n, že dĺžka dlhšej strany bitmapy bude aspoň 2^n. Vtedy Hilbertova krivka generovaná postupnosťou smerov $D(n)$ navštívi všetky pixely bitmapy.

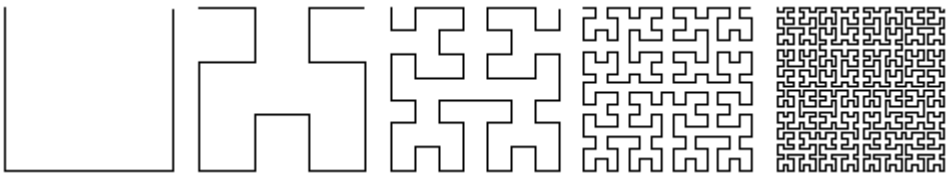
Peanova krivka

Ďalšou priestor vypĺňajúcou krivkou, ktorú môžeme použiť, je Peanova krivka. Túto krivku vymyslel v roku 1890 pán Giussepe Peano [6]. Podobne ako pri Hilbertovej krivke, aj táto sa dá definovať pomocou špeciálneho očísľovania štvorcov a ich delenia, tentokrát až na deväť častí. Krok by sme tentokrát zmenšovali o tretinu. My ale potrebujeme mať konštatný krok a presnú postupnosť smerov. Preto ju definujeme podobne ako Hilbertovu krivku pomocou rekurzívnych funkcii.

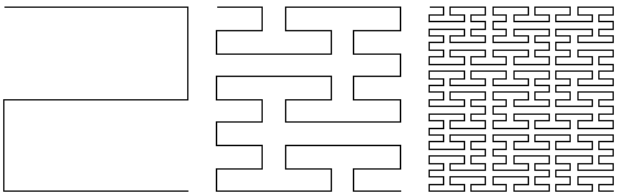
$$
\begin{aligned}
Zd(n) &= Zd(n-1) \rightarrow Su(n-1) \rightarrow Zd(n-1) \downarrow Sd(n-1) \leftarrow Zu(n-1) \leftarrow Sd(n-1) \downarrow \\
&\quad Zd(n-1) \rightarrow Su(n-1) \rightarrow Zd(n-1) \\
Zu(n) &= Zu(n-1) \leftarrow Sd(n-1) \leftarrow Zu(n-1) \uparrow Su(n-1) \rightarrow Zd(n-1) \rightarrow Su(n-1) \uparrow \\
&\quad Zu(n-1) \leftarrow Sd(n-1) \leftarrow Zu(n-1) \\
Su(n) &= Su(n-1) \rightarrow Zd(n-1) \rightarrow Su(n-1) \uparrow Zu(n-1) \leftarrow Sd(n-1) \leftarrow Zu(n-1) \uparrow \\
&\quad Su(n-1) \rightarrow Zd(n-1) \rightarrow Su(n-1) \\
Sd(n) &= Sd(n-1) \leftarrow Zu(n-1) \leftarrow Zd(n-1) \downarrow Zd(n-1) \rightarrow Su(n-1) \rightarrow Zd(n-1) \downarrow \\
&\quad Sd(n-1) \leftarrow Zu(n-1) \leftarrow Sd(n-1) \\
Zd(0) &= Zu(0) = Su(0) = Sd(0) = []
\end{aligned}
$$

Generovať postupnosť začíname volaním $Zd(n)$. Prvé tri iterácie pre $n = 1, 2, 3$ môžete vidieť na obrázku 4.3.

Strana štvorca, ktorý vypĺňa peanova hrivka rastie mocninou 3. Preto Opäť treba nájsť také n, že $Zd(n)$ vygeneruje krivku,ktorá má stranu aspoň takú dlhú ako je dlhšia strana bitmapy.



Obr. 4.2: Hilbertova krivka



Obr. 4.3: Penaova krivka

Algoritmus

Teraz uvedieme algoritmus, podľa ktorého priraďujeme priadze krížikom. Opäť budeme mať paletu P ako inštanciu triedy `ColorPalette`, bitmapu B ako dvojrozmerné pole a výstupnú predlohu R ako inštanciu triedy `Pattern`. Volanie `next()` vráti vporadí ďalší bod priestor vypĺňajúcej krivky.

```
1  x = 0
2  y = 0
3  kym existuje nenavstiveny pixel:
4        id = P.getNearestIDToColor(B[y][x])
5        R.setSquare(x,y,id)
6        error = B[y][x] - P.getColor(id)
7        x,y = next()
8        B[y][x] = B[y][x] + error
```

Samozrejme, aj pri tomto riešení je ešte treba ošetriť to, či je vygenerovaný bod v bitmape.

Výstupy tohto postupu si môžete pozrieť na obrázkoch A.8e, A.9e a A.10e.

Môžete si všimnúť, že výstupy sú veľmi zrnité. Tento problém riešil aj pán Riemersma. V [7] navrhol úpravu tohto postupu tak, aby sa neprenášala celá chyba naakumulovaná zo všetkých predchádzajúcich pixelov, ale len z niekoľkých posledných. Malo by ich byť aspoň 16. Táto chyba je navyše odstupňovaná tak, že najviac chyby sa prenesie z posledného spracovaného pixela a najmenej z najskôr spracovaného pixela, ktorý ešte patrí do postupnosti. Nech E je zoznam chýb pre posledných d pixelov indexovaný od nuly. Potom chyba ktorá sa pripočíta nasledujúcemu pixlu je

$$\sum_{i=0}^{d-1} \frac{E[i]}{d} exp\left(\frac{ln(d)}{d-1} i\right) \tag{4.1}$$

Upravený algoritmus bude nasledovný:

```
1  x = 0
2  y = 0
3  E = zoznam dlzky d naplneny nulami
4  kym existuje nenavstiveny pixel:
5        error = 0
6        pre i od 0 po d-1:
7            error = error + E[i]*exp(log(d)*i/(d-1))/d
8        id = P.getNearestIDToColor(B[y][x]+error)
9        R.setSquare(x,y,id)
10       vyhod nulty prvok zoznamu
11       pridaj na koniec B[y][x] - P.getColor(id)
12       x,y = next()
```

Výstupy tohto postupu si môžete pozrieť na obrázkoch A.8f, A.9f a A.10f.

Podľa Riemersmu je najvhodnejšie vo všeobecnom prípade použiť Hilbertovu krivku [7]. Dôvodom je jej časté menenie smeru. My sme sa však rozhodli použiť aj Peanovu krivku. Medzi týmito krivkami nemáme favorita. Obe krivky majú približne rovnaké výsledky, čo sa týka našich kritérii.

Kapitola 5

Implementácia

Aplikáciu sme písali v jazyku Python 3. Na prácu s farbami, obrázkami a na grafické rozhranie sme použili knižnicu PySide. Celá aplikácia je v nasledujúcich súboroch.

- `CalculatingMethods.py`
- `ColorPalette.py`
- `Constrants.py`
- `LeftWidget.py`
- `Miniature.py`
- `Needle.py`
- `Pattern.py`
- `PatternLabel.py`
- `PatternWidget.py`
- `SettingsWidget.py`
- `UsedColors.py`

Až na dve výnimky (`CalculatingMethods.py` a `Constrants.py`) názvy súborov zodpovedajú názvom tried v nich uložených. Okrem týchto zdrojových kódov implementácia obsahuje:

- priečinok so záznamami o farebných paletách jednotlivých značiek vyšívacích priadzí
- priečinok obsahujúci všetky symboly

Implementácia je dostupná na adrese `http://people.ksp.sk/ andulka/needle.php`

5.1 Implementácia riešení

Všetky implementácie algoritmov popísaných v kapitolách 2 až 4 sme sústredili do súboru `CalculatingMethods.py`.

Výber palety

Pre každé riešenie sme implementovali samostatnú funkciu. Vstupom pre tieto funcie je bitmapa, paleta priadzí a maximálny počet priadzí, ktoré môžeme vybrať. Výstupom je inštancia triedy `ColorPalette`. Keďže vytváranie štatistiky bolo treba pri každom riešení, vyrobili sme na to špeciálnu funkciu.

Zmena veľkosti bitmapy

Pre obe riešenia sme vytvorili samostatné funkcie. Vstupom pre priemerovanie je bitmapa, dĺžka δ spomínaná v tretej kapitole, z ktorej sa neskôr vypočítajú rozmery výstupnej bitmapy. V prípade detekcie hrán ešte pribudol parameter k. Podľa jeho znamienka sa určí ktorú stranu hrany chce používateľ uprednostniť a jeho absolútna hodnota určí ako veľmi. Nulu sme špeciálne ošetrili. Výstupom je bitmapa v želaných rozmeroch.

Priradenie priadzí krížikom

Pre nahradenie najbližším a error diffusion dithering sme vytvorili samostatné funkcie. Nahradenie najbližším dostane ako vstup iba paletu (inštanciu triedy `ColorPalette`) a bitmapu. V prípade error diffusion ditheringu medzi vstupy patrí aj filter zadaný v tvare dvojrozmerného poľa. Značkou pre práve spracovávaný pixel vo filtri je -1.Výstupom je predloha (inštancia triedy `Pattern`).

Pri Riemersma ditheringu sme vytvorili dve triedy. Jednu pre pôvodné riešenie a druhú pre to upravené. Aby sme nemuseli robiť špeciálnu triedu pre každú priestor vypĺňajúcu krivku, zaviedli sme špeciálny spôsob zápisu rekurzívnych funkcií, ktoré sme uviedli v štvrtej kapitole. Obe triedy pri vytváraní dostanú bitmapu, paletu (inštanciu triedy `ColorPalette`), základňu podľa ktorej rastie štvorec, ktorý vyplní jedna z našich dvoch priestor vypĺňajúcich kriviek, pravidlá vytvárania krivky a pri upravenom riešení ešte aj dĺžku zoznamu posledných chýb. Hneď po vytvorení sa zároveň vypočíta predloha. Na jej získanie treba zavolať funkciu triedy `getPattern()`.

5.2 Používateľské rozhranie

Grafická knižnica PySide má pre všetky grafické prvky ako sú talčidlá, posúvače atď. jedného spoločného predka, triedu `QWidget`. Navyše sa táto trieda dá použiť ako samostatný grafický prvok. Zobrazuje sa ako šedá prázdna plocha. Na tento widget je potom možno dávať ďalšie widgety, alebo naňho kresliť. Všetky triedy, ktoré budú nasledovať sú potomkami triedy `QWidget`.

Rozlišujeme dva typy predlohy. Vlastnú predlohu a predlohu podľa bitmapy. Pri oboch predlohách vyzerá aplikácia veľmi podne, ale má isté odlišnosti. Preto sme pri implementácií tried použili návrhový vzor Abstract Factory. Všetky triedy, ktoré budú začínať `Image` patria spolu a všetky tie, ktoré budú začínať `Own` patria tiež spolu. Je tam aj množina tried, ktoré nemajú ani jednu takúto predponu. V tom prípade sú to spoločné triedy pre oba typy predlohy, alebo špecifické pre jednu triedu.

Trieda `PatternLabel`

Slúži na zobrazenie predlohy, ale namiesto symbolov zobrazuje priamo farby vyšívacích priadzí. Tak používateľ vie, ako bude výšivka zhruba vyzerať. Táto trieda taktiež umožňuje označovať jednotlivé štvorčeky.

Trieda `Miniature`

Táto trieda slúži na zobrazenie miniatúry vstupného obrázku, pri generovaní predlohy podľa bitmapy. Taktiež vyznačuje oblasť vstupnej bitmapy, ktorá zodpovedá viditeľnej časti predlohy v `PatternLabel`.

Trieda `UsedPalette`

Zobrazuje v sebe všetky použité priadze, ktoré sú v predlohe. Priadze sú označené číslom a značou. Ku každej priadzi je ďalej uvedená je farba, priradený symbol ,počet krížikov k nej prislúchajúcich a počet pradien. Ďalej umožňuje vymeniť symbol pre ľubovoľnú priadzu a dovoľuje ju nastaviť ako aktuálnu priadzu.

Triedy s príponou `LeftWidget`

`LeftWidget` je pre každý typ predlohy iný. Preto sme definovali `AbstractLeftWidget`, ktorý je nadtriedou pre `ImageLeftWidget` a `OwnLeftWidget`. V `ImageLeftWidget` sa nachádza inštancia triedy `Miniature`. V oboch sa ďalej nachádzajú dve tlačidlá a inštancia triedy `UsedPalette`. Tlačidlá slúžia na editovanie predlohy. Jedným tlačidlom sa nastaví

aktuálna priadza, ale dá sa nastaviť aj z `UsedPalette`. Druhým tlačidom potom aplikujeme túto priadzu na označené štvorčeky.

Triedy s príponou `SettingsWidget`

Tieto triedy slúžia na zozbieranie nastavení od užívateľa.

V prípade `ImageSettingsWidget` sú to nastavenia výšky alebo šírky, značky priadze, maximálneho počtu priadzí, ako budeme vyberať paletu, ako veľmi a ktorú stranu hrany uprednostníme a nakoniec akú metódu použijeme pri priraďovaní priadzí krížikom. Keď užívateľ stlačí tlačidlo `Do`, tak sa tieto nastavenia posunú kompetentnejšej triede.

V prípade `OwnSettingsWidget` používateľ nastavuje iba rozmery výšivky. Po stlačení tlačidla `Do` sa opäť tieto parametre posunú kompetentenjšej triede.

Triedy s príponou `PatternWidget`

Triedy s touto príponou sú najkompetentnejšie. Do tejto skupiny spadajú triedy `AbstractPatternWidget`, `ImagePatternWidget` a `OwnPatternWidget`, pričom prvá je nadtriedou zvyšných dvoch.

V triede `AbstractPatternWidget` sú definované spoločné funkcie a spoločný vzhľad. `PatternWidget` je rozdelený na tri časti. Vrchnú časť a dve spodné časti. Vrchná časť sa potom delí na nastavenia, editovaciu časť a obsluhu zobrazovania predlohy. Nastavenia sú konkretizované v potomkoch. Je to buď `ImageSettingsWidget` alebo `OwnSettingsWidget`, podľa toho, akého typu je potomok. Editovacia časť pozostáva z dvoch tlačidiel, ktoré umožňujú vracať a robiť znovu zmeny v predlohe. Obsluha zobrazovania predlohy pozostáva z posúvača, ktorý určuje priblíženie predlohy. V ľavej spodnej časti je potom inštancia niektorej z tried s príponou `LeftWidget`, podľa typu triedy z tejto skupiny. V pravej spodnej časti sa nachádza inštancia triedy `PatternLabel` pre všetky triedy tejto skupiny.

Spoločná funkcionalita zahŕňa exportovanie, nastavenie priblíženia a celú réžiu okolo editovania predlohy. V triedach `ImagePatternWidget` a `OwnPatternWidget` sú definované len funkcie na ukladanie, otvrátanie a generovanie predlohy. A v prípade `ImagePatternWidget` je nevyše zabezpečená komunikácia medzi trieda `Miniature` a `PatternLabel`, aby sa správne zobrazovala vyznačená oblasť.

Trieda `Needle`

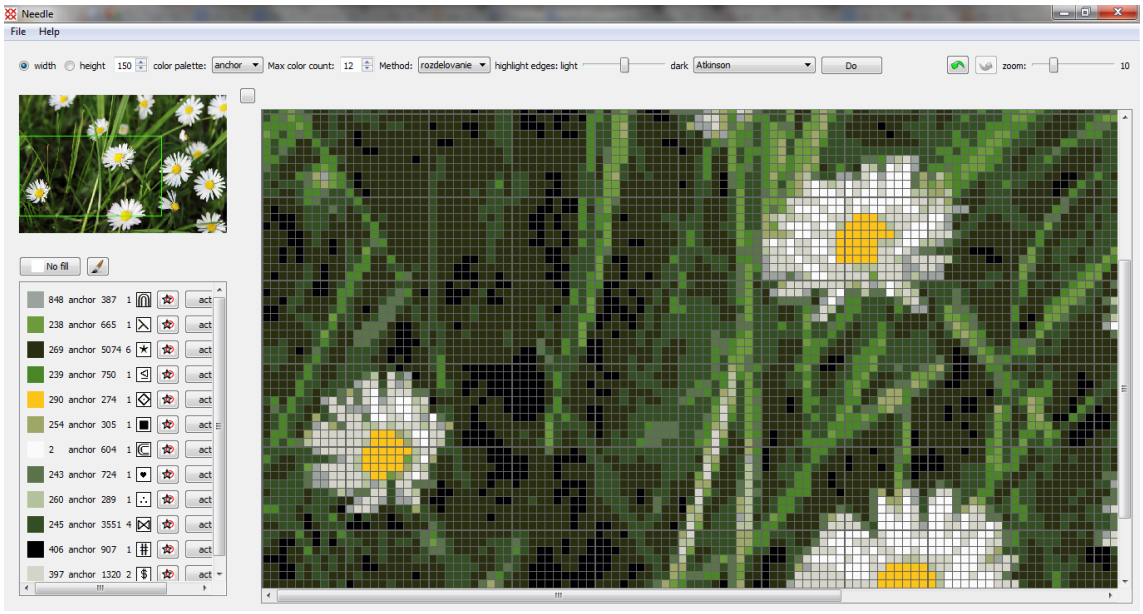
Trieda `Needle` tvorí hlavné okno aplikácie. Spravuje menu a vytvára správny typ `PatternWidgetu`.

V menu si používateľ môže zvoliť, či chce generovať predlohu podľa bitmapy, alebo si vytvorí vlastnú. Podľa toho, čo si používateľ vyberie, `Needle` vyrobí inštanciu príslušnej triedy. Pri výbere predlohy podľa bitmapy sa používateľovi hneď otvorí dialógové okno, ktorým si vyberie bitmapu.

Ďalej môže zvoliť otvorenie predlohy. Súbor, ktorý slúži na úschovu predlohy má hneď na prvom riadku identifikátor typu predlohy. Podľa tohto identifikátora `Needle` vytvorí inštanciu príslušnej triedy `PatternWidgetu` a zavolá jeho funkciu na načítanie zo súboru.

Užívateľ ďalej v menu nájde ukladanie a exportovanie. V oboch prípadoch sa zavolá príslušná funkcia `PatternWidgetu`, ale exportovanie je umožnené až potom, čo užívateľ svoju predlohu uloží.

Ukážku používateľského rozhrania môžete vidieť na obrázku 5.1



Obr. 5.1: Ukážka používateľského rozhrania

Záver

V práci sme navrhli vlastné riešenia problému výberu palety a zmeny veľkosti vstupnej bitmapy. Ďalej sme preskúmali vhodnosť rôznych typov ditheringu, ako riešenia problému priradenia priadzí krížikom. Všetky tieto riešenia sme implementovali a vytvorili k nim používateľské prostredie.

Výstupy z našej aplikácie sme porovnali s výstupmi voľne dostupnej aplikácie Stitch Art Easy! 4.0, ktorá slúži rovnakému účelu ako naša aplikácia. Oproti tejto aplikácii sme zaznamenali lešie výsledky pri výbere palety. Rozdiely sú viditeľné na prvý pohľad na obrázku A.11. Je vidieť, že naša aplikácia vybrala oveľa jasnejšie farby, ktoré viac zodpovedajú vstupnému obrázku. Pri tomto teste sme použili rovnaké priradenie farieb vyšívacím priadzam ako v Stitch Art Easy! 4.0.

Táto aplikácia dokonca ponúka možnosť použiť dithering. Pri testovaní sme si všimli, že ich dithering vytvára podobné vzory ako error diffusion dithering s filtrom Floyd-Steinberg. Naša aplikácia ponúka niekoľko druhov ditheringov, ktoré si môže používateľ vybrať.

Naša aplikácia však zaostáva v rýchlosti generovania predlohy. Tento nedostatok sa však dá zčasti odtrániť tým, že použijeme paralelné procesy. Mnohé z algoritmov, ktoré sme v práci spomenuli sú vysoko paralelizovateľné.

Pri testovaní sme si všimli ešte jeden problém. Naše programi nemali zhodné farby pre jednotlivé priadze. Rozdiely neboli veľké, ale boli viditeľné. Takisto sa nám stalo, že pri vyšívaní podľa vygenerovanej predlohy mali priadze vybraté programom a reálne farby medzi sebou iné relatívne vzdialenosti. Priadze, ktoré mali v počítači celkom jasne odlišné farby, boli v reáli takmer neodlíšiteľné. Preto vyvstáva otázka, ako zdigitalizovať farby priadzí tak, aby čo najviac zodpovedali realite.

Chceli by sme v našej aplikácii ešte vylepšiť výber palety. Zatiaľ sú všetky naše algoritmy greedy. Navyše sme napevno určili koeficient, ktorý je pre väčšinu obrázkov dobrý, ale určite nie je optimálny. Mohlo by byť však zaujímavé, keby sme hľadali najlepšiu paletu vzhľadom na koeficient podobnosti. Možností je priveľa na to, aby sme všetky vyskúšali. Preto by sa možno na hľadanie nejlepšej palety dala použiť metóda hill climbing.

Dodatok A

Výstupy

V tejto prílohe ukazujeme výstupy algoritmov spomínaných v kapitole 2 až 4. Na testovanie sme použili bitmapy na obrázku A.1. Bitmapu A.1c nakreslila moja dlhoročná kamarátka Lucia Petríková. Bitmapu A.1a môžete nájsť na adrese:
`http://pixabay.com/cs/barevné-červená-modrá-žlutá-266992/`
a bitmapu A.1b môžete nájsť na adrese:
`http://pixabay.com/cs/sedmikráska-zahrada-louka-květiny-111892/`.

Pri testovaní budeme používa vždy paletu vyšívacích priadzí značky Anchor.

(a) Bez farebných skokov

(b) Fotografia

(c) Kresba

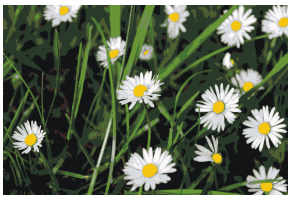
Obr. A.1: Testovacie bitmapy

riešenie	**koef. k**	**koef. podob.**	**odstr.-2. fáza**	**odstr.-3. fáza**	**paleta**	**upravená bitm.**
1	-	302090691	240	-		
2	0.005	192954378	233	7		
2	0.01	302090691	240	0		
2	0.015	302090691	240	0		
3	0.005	191008648	233	7		
3	0.01	288404557	240	0		
3	0.015	288404557	240	0		

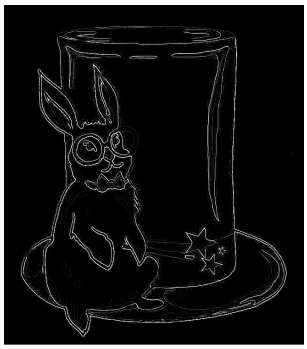
Obr. A.2: Vstupná bitmapa: A.1c, maximum priadzí: 12

riešenie	**koef. k**	**koef. podob.**	**odstr.-2. fáza**	**odstr.-3. fáza**	**paleta**	**upravená bitm.**
1	-	1036817468	193	-		
2	0.005	952448445	143	50		
2	0.01	865422735	170	23		
2	0.015	869307342	179	14		
3	0.005	963681154	136	57		
3	0.01	887425492	168	25		
3	0.015	865015393	175	18		

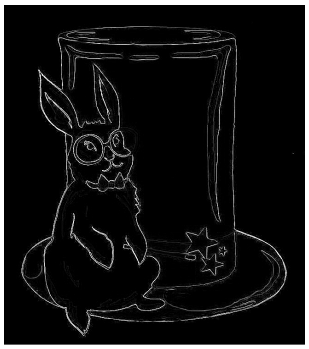
Obr. A.3: Vstupná bitmapa: A.1a, maximum priadzí: 18

riešenie	koef. k	koef. podob.	odstr.-2. fáza	odstr.-3. fáza	paleta	upravená bitm.
1	-	280846204	201	-		
2	0.005	153328352	181	20		
2	0.01	155688201	192	9		
2	0.015	151924883	195	6		
3	0.005	145904448	181	20		
3	0.01	144986676	190	11		
3	0.015	144769780	195	6		

Obr. A.4: Vstupná bitmapa: A.1b, maximum priadzí: 12



(a) Svetlá strana hrany



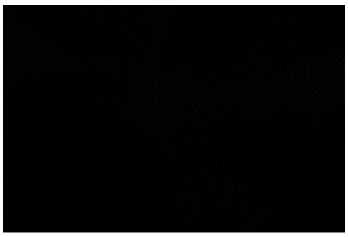
(b) Tmavá strana hrany

(c) k = -5

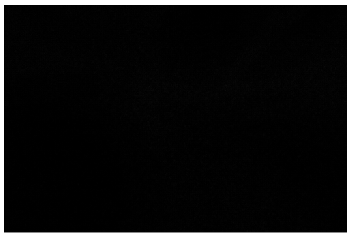
(d) k = 0

(e) k = 5

Obr. A.5: Edge detection a zmena veľkosti bitmapy A.1c na sirku 120 pixelov



(a) Svetlá strana hrany



(b) Tmavá strana hrany

(c) k = -5

(d) k = 0

(e) k = 5

Obr. A.6: Edge detection a zmena veľkosti bitmapy A.1a na šírku 150 pixelov

(a) Svetlá strana hrany

(b) Tmavá strana hrany

(c) k = -5

(d) k = 0

(e) k = 5

Obr. A.7: Edge detection a zmena veľkosti bitmapy A.1b na šírku 150 pixelov

(a) Bez ditheringu

(b) Floyd-Steinberg

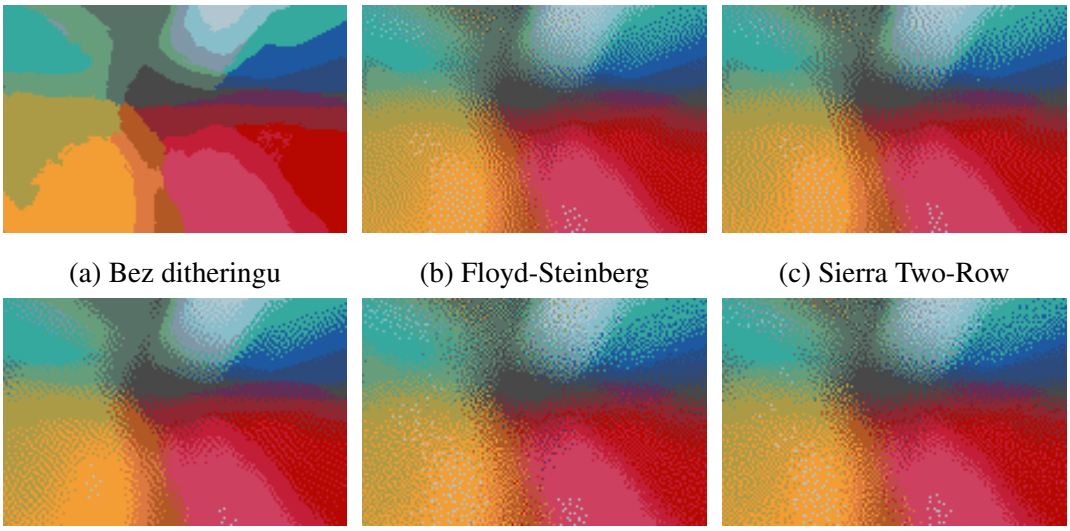
(c) Sierra Two-Row

(d) Atkinson

(e) Riemersma+Hilbert neupr.

(f) Riemersma+Hilbert upr.

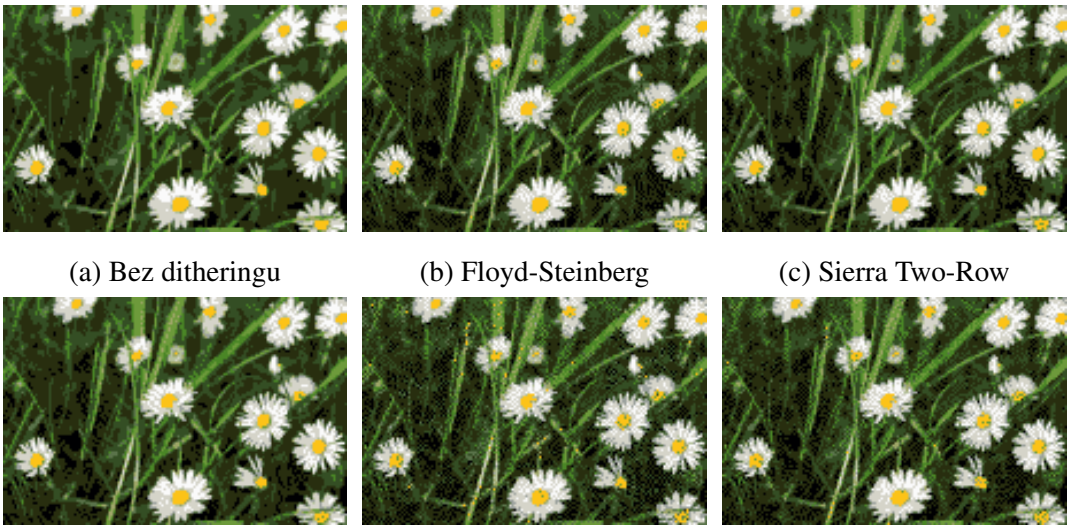
Obr. A.8: Dithering: vstupná bitmapa A.1c, max. priadzí 12, šírka 120, 3. riešenie va výber palety s k = 0.005, uprednostnenie tmavej strany hrán



(a) Bez ditheringu (b) Floyd-Steinberg (c) Sierra Two-Row

(d) Atkinson (e) Riemersma+Hilbert neupr. (f) Riemersma+Hilbert upr.

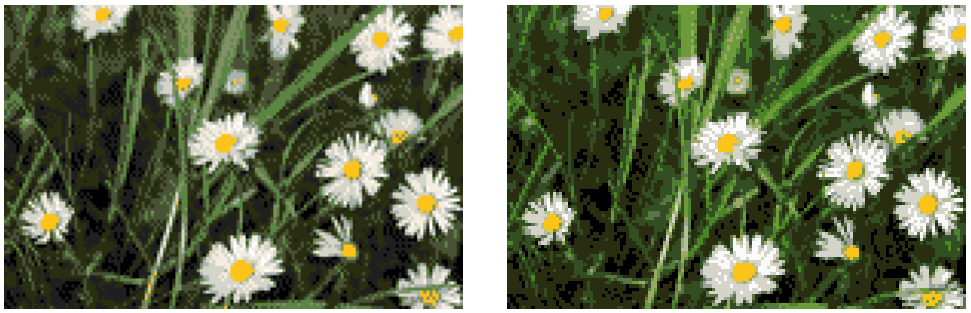
Obr. A.9: Dithering: vstupná bitmapa A.1a, max. priadzí 18, šírka 150, 2. riešenie na výber palety s k = 0.01, priemerovanie



(a) Bez ditheringu (b) Floyd-Steinberg (c) Sierra Two-Row

(d) Atkinson (e) Riemersma+Hilbert neupr. (f) Riemersma+Hilbert upr.

Obr. A.10: Dithering: vstupná bitmapa A.1b, max. priadzí 12, šírka 150, 3. riešenie na výber palety s k = 0.01, priemerovanie



(a) Stitch Art Easy! 4.0 (b) Needle

Obr. A.11: Porovnanie výstupov nášho programu s konkurečným pregramom Stitch Art Easy! 4.0, zadanie: šírka 150 krížikov, paleta Anchor, maximálne 12 priadzí a vstupná bitmapa A.1b

Literatúra

[1] D. Burkes. Presentation of the burkes error filter for use in preparing continuous-tone images for presentation on bi-level devices. LIB 15 (Publications), CIS Graphics Support Forum, 1988.

[2] R. W. Floyd and L. Steinberg. An adaptive algorithm for spatial gray scale. *SID 1975, International Symposium Digest of Technical Papers*, vol 1975m, 1975.

[3] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Prentice Hall, Upper Saddle River, New Jersey 07458, 2 edition, 2002.

[4] D. Hilbert. Ueber die stetige abbildung einer linie auf ein flächenstück. *Mathematische Annalen*, 38:459–460, 1891.

[5] J. Jarvis, C. Judice, and W. Ninke. A survey of techniques for the display of continuous tone pictures on bilevel displays. *Computer Graphics and Image Processing 5*, vol. 5, 1976.

[6] G. Peano. Sur une courbe, qui remplit une aire plane. *Mathematische Annalen*, 36:157–160, 1890.

[7] T. Riemersma. A balanced dither algorithm. *C/C++ Users Journal*, vol. 16, issue 12 1998.

[8] F. Sierra. LIB 17 (Developer's Den), CIS Graphics Support Forum.

[9] P. Stucki. Mecca - a multiple error correcting computation algorithm for bi-level image hard copy reproduction. research report rz1060, IBM Research Laboratory, Zurich,Switzerland, 1981.

[10] R. Ulichney. *Digital Halftoning*. The MIT Press, Cambridge, 1987.