Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra informatiky a výpočetní techniky

Bakalářská práce

Využití shlukování pro digitalizované obrazy

Plzeň, 2009 Pavel Hulej

Zde bude originál zadání

Prohlášení

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů.

. . . . . .
Pavel Hulej

V Plzni dne 13. 5. 2009

Poděkování

Na prvním místě bych chtěl poděkovat Ing Jiřímu Skálovi nejenom za jeho odbornou pomoc při zpracování práce, ale také za ochotu pomoci s problémy nejenom týkajícími se práce. Další velký dík patří Doc. Dr. Ing. Ivaně Kolingerové za odbornou pomoc při psaní této práce.

Dále bych rád poděkoval svým rodičům, kteří mně vytvářeli příjemné zázemí a poskytovali maximální podporu ve studiu.

Abstrakt

Cílem této práce je úprava a urychlení již existujícího programového vybavení pro shlukování bodů a jeho aplikace na digit. obraz. Nejprve bylo však nutné knihovnu přepracovat z použití na geometrických modelech do použití pro digitalizované obrazy. Poté vyzkoušet, jak se shlukovací knihovna chová na obrázcích a posoudit, zdali je pro ně vhodná. Protože původní verze knihovny nesplňovala požadavky kladené na rychlost, bylo potřeba se zaměřit z velké části na urychlení tohoto algoritmu. A na závěr vyzkoušet spojování podobných shluků.

This work aims at modifications an speed-up of already existing library for clustering of points and its application to digital images. At first it was necessary to modify the software to by able to process digital images. Then to check out how clustering library behaves on images and to explore if it is acceptable for them. As the original version of library did not not fulfil the requirements on speed, therefore, it was necessary to concentrate, mainly on speed-up of the used clustering algorithm. And, at the end, test out joining of similar clusters.

Obsah:

1. Úvod

Ing. Jiří Skála vypracoval v roce 2008 knihovnu pro shlukování data streamů. Aplikace původní práce spočívala v manipulaci s geometrickými modely. To se provádí pomocí shluků. Pro velké objemy dat byl použit víceúrovňový model, kde bylo cílem dostat nejvyšší úrovně dat do vnitřní paměti.

Mým úkolem bylo použít tuto knihovnu a aplikovat ji na digitalizované obrázky, Posoudit zda je shlukování vhodné pro digit. obrázky, a dále pak získat vhodnou reprezentaci dat pro kompresi. Knihovnu urychlit a umožnit spojování podobných shluků.

Nejprve bylo potřeba knihovnu upravit. Původně sloužila pro zpracování bodů ve 3D. Nové použití vyžaduje shlukování pixelů, které mají souřadnice x, y a barvu R, G, B. Dále pak najít přepočet barvy a vzdálenosti dvou bodů. Poté vyzkoušet na mnoha obrázcích, jak se shlukovací knihovna chová, a dát doporučení pro parametry, se kterými ji můžeme nastavovat. Celkově ji urychlit změnou výpočtu počátečního řešení, použitím mřížky a s použitím vhodně zvolených bodů obrázku. Dále pak urychlit s použitím menšího počtu iterací, vyzkoušet její chování na jiných metrických prostorech. Použití manhattanské a 2. mocniny euklidovské metriky bez odmocniny. A na závěr umožnit spojování shluků s podobnou barvou.

1.1 Rozvržení práce

V teoretické části se budeme zabývat :

- Jednoduchými shlukovacími metodami, které jsou obecně známé.
- Popisem knihovny Ing. Jiřího Skály.
- Novými částmi, které jsme potřebovali do knihovny přidat.

V praktické části:

- Urychlováním pomocí snižování iterací a jiného výpočtu počátečního řešení.
- Vlivem snížení počtu iterací na obrázek.
- Změnami metrik.
- Nastavením správných parametrů pro digit. obrázek.
- Spojováním podobných shluků.

2. Teoretická část

Na začátku této kapitoly je třeba se zmínit, že technik pro redukci barev, segmentaci a kompresi digitalizovaných obrazů existuje mnoho. My se zde pouze zmíníme o některých nejjednodušších a nejznámějších. Pak se již budeme věnovat shlukovací knihovně Ing. Jiřího Skály.

2.1 Prahování

Prahování obrazu provádí funkce která upravuje jasové či barevné složky pixelů obrazu podle následujícího vztahu 2.1 (převzatého z [1])

$$f(o) = \begin{cases} a & pokud \quad o < práh \\ b & pokud \quad o \geq práh \end{cases} \quad \textbf{(2.1)}$$

kde

o vstupní hodnota jasu nebo barvy.
f(o) výsledná hodnota.
a nová hodnota pro vstupní hodnotu *o* pod prahem, často bývá bílá
b nová hodnota pro vstupní hodnotu *o* nad prahem, většinou je černá

Tuto metodu lze použít k detekci hran a rozpoznávaní znaků.

2. 2 Kvantizace barev

Je to proces, který snižuje počet barev použitých v obrázku, obvykle s úmyslem takovým, že nový obrázek by měl být vizuálně podobný původnímu obrázku. Kvantizace je rozhodující pro zobrazení obrázků s mnoha barvami na zařízeních, která nemohou zobrazovat tolik barev, nebo kvůli paměťovým omezením, a umožňují efektivní komprese některých obrázků, např. GIF nebo PNG.

Většina standardních metod řeší kvantizaci jako problém seskupení bodů v třírozměrném prostoru, kde body představují barvy zjištěné v původním obrazu, na třech osách jsou zobrazeny tři barevné kanály. Téměř každý třídimenzionální shlukovací algoritmus může být použit pro kvantizace. Najdeme seskupení bodů ve shluku. Vezme se průměr barev, které jsou vázány na shluk. Obvykle se používá barevný prostor RGB, ale může se použít Lab. Více o kvantizaci je napsáno ve [2].

2. 3 Segmentace

Vztahuje se k procesu rozdělování digitálního obrazu v různých oblastech nebo (souborech pixelů) na části. Cílem segmentace je zjednodušit anebo změnit reprezentaci obrazu v nějakou lepší strukturu. Segmentace se zpravidla používá k lokalizaci objektů a hranic (linie, křivky atd.) v obrazech. Více o segmentaci se dovíme ve [3].

Nástroje segmentace - existuje jich velmi mnoho, proto si uvedeme pouze dva:

Histogramy

Histogram je reprezentací rozložení barev v obrazu. Používá se v počítačové grafice a digitální fotografii. Vyjadřuje poměrné zastoupení počtu pixelů každého z daných barevných rozsahů v buď dvourozměrném (2D) nebo třírozměrném (3D) barevném modelu.

Histogram je měřítkem statistického popisu frekvenčního rozložení ve vztahu k výskytu frekvencí různých tříd barev. Třídy barev jsou určité oblasti v barevném prostoru. Každá třída barev v obrazu má svůj index, kterým se při výpočtu histogramu násobí. Více o histogramech se dozvíme [4].

Metody založené na histogramech jsou velmi účinné při srovnání s jinými metodami segmentace obrazu, protože obvykle vyžadují pouze jeden průchod pixelů. V této technice je histogram počítán ze všech pixelů v obrázku a vrcholy v histogramu jsou použity k nalezení

shluků v obrazu. Barvy nebo intenzity mohou být použity jako rozsah. Více o jejich použití se dozvíme v [3].

Problém barevné redukce vypadá jako shlukovací problém. Například pro obrázek převedený do 16 barev je potřeba najít optimální pozici 16 shluků v barevném RGB prostoru, který je reprezentován vstupním barevným prostorem. Chybu toho prostoru se snažíme minimalizovat. K tomuto účelu obvykle používáme tzv. střední kvadratickou chybu (rozptyl).

Nejjednodušší cestou, jak získat tyto shluky, je redukce posledních několika bitů složek RGB. Tento velmi jednoduchý a rychlý algoritmus je však pro většinu aplikací nepřijatelný.

Jiný běžný algoritmus pro barevnou redukci je výběr mediánu, další používané algoritmy jsou barevné neuronové kvantizační algoritmy. Tyto algoritmy se liší ve výstupní kvalitě, implementační obtížnosti, paměťové náročnosti a délky běhu výpočtu. Je jasné, že neexistuje „nejlepší“ algoritmus pro barevnou redukci. Další z jednoduchých algoritmů pro shlukování je klasický K-Means algoritmus.

K-means algoritmus:

1. Vybereme K bodů (středů) v prostoru
2. Přiřadíme každý vstupní vektor k nejbližšímu středu.
3. Přepočítáme pozice všech středů tak, že nová pozice každého středu bude průměrem všech vektorů obsažených v tomto shluku.
4. Jdeme zpět na krok, 2 dokud nejsou pozice středů dlouho v klidu.

Část, která není specifikována, je cesta, jak změnit startovací středy. Často používáme náhodný výběr vektorů.

Implementace originálního K-means shlukování je velmi přímá. Předpokládejme redukci do K barev: Paleta je ustanovena z K náhodných pixelů ze vstupního obrazu. Potom se užije K-means algoritmus k modifikaci této palety. A nakonec je vstupní obrázek převeden na výstupní. Další implementační detaily a podrobnosti se dovíme v [5].

3. Popis řešení

Naše řešení umožňuje provádět shlukování dig. obrazů pomocí výpočtu rozdílu barvy RGB barevného prostoru a dále pak pomocí kombinace vzdálenosti bodů a barvy, čímž můžeme ovlivňovat výsledek shlukovaní.

Další snahou je urychlení stávající knihovny a umožnění jejího běhu pro velké obrázky, to se provádí pomocí snížení počtu iterací, změnou generování počátečního řešení a nakonec změnou metrického systému.

Na závěr pro účely komprese umožnit spojování shluků s podobnou barvou a výsledek uložit do obrázku v podobě „Masky“. Masku tvoří hranice shluků, které jsou bílou barvou, ostatní plochy jsou černě.

3.1 Výpočet rozdílu barvy v RGB

Abychom mohli v barvách porovnávat vzdálenosti, je potřeba spočítat rozdíl v barvě. K tomu ji však potřebuje rozložit na jednotlivé složky *(R,G,B)*. Pokud se nám toto podaří, můžeme, se pustit do výpočtu rozdílu barvy. Při výpočtu se snažíme dodržet citlivost lidského oka, které je citlivější na zelenou barvu a naopak méně citlivé na modrou barvu, jak ukazuje následující vztah 3.1, který počítá rozdíl barvy dvou pixelů s ohledem na vlastnosti lidského zraku.

$$d_{rgb} = 0.3 \cdot |r_1 - r_2| + 0.6 \cdot |g_1 - g_2| + 0.1 \cdot |b_1 - b_2| \qquad \textbf{(3.1)}$$

kde:

d_{rgb}	*celkový rozdíl v barvě*
$r_1 - r_2$	*rozdíl mezi dvěma sousedními pixely v červené barvě*
$g_1 - g_2$	*rozdíl mezi dvěma sousedními pixely v zelené barvě*
$b_1 - b_2$	*rozdíl mezi dvěma sousedními pixely v modré barvě*

3.2 Kombinace vzdálenosti bodů a barvy

Protože nelze jen tak míchat vzdálenost bodů (pixelů) spolu se vzdáleností v barvě (viz vztah 3.1), je třeba zavést nějaký přepočet. Uděláme to tak, že rozdílu v barvě přiřadíme určitou váhu. Tím můžeme v „zesilovat“ nebo naopak „mírnit“ význam barvy. To ukazuje vztah 3.2, kde se vzdálenost počítá ze souřadnic i barvy pixelů

$$d = \sqrt{[dx^2 + dy^2 + (vahaJasu \cdot d_{rgb})^2]} \qquad \textbf{(3.2)}$$

kde:

d	*výsledná kombinace vzdálenosti a barvy, tedy výsledná vzdálenost*
dx	*rozdíl v x-ové souřadnici tj. dx =* x_1 *-* x_2*.*
dy	*rozdíl v y-ové souřadnici tj. dy =* y_1 *-* y_2
vahaJasu	*koeficient váhy barvy.*
d_{rgb}	*celkový rozdíl v barvě.*

3.3 Shlukování

Výhodou toho řešení Ing. Jiřího Skály (pro manipulaci s geometrickými objekty) je, že nepotřebuje zadávat počet shluků nebo počet barev jako např. u K-means algoritmu. Dále počítáme nejenom s barvou, ale také se vzdáleností bodů. Výsledky lze též ovlivňovat nastavením parametrů váhy jasu a velikostí shluků.

Pro konstrukci shlukování byla použita knihovna od Ing. Jiřího Skály. Je napsána

v programovacím jazyce C#. Knihovna řeší shlukování jako tzv. problém facility[1] location. Obecná formulace zní takto: Je dána množina potenciálních středů shluků (tzv. facility) a množina klientů. Každý klient by měl být připojen k jedné otevřené facility. Za otevření každé facility je nutno zaplatit cenu *fc* (facility cost); za připojení klienta je nutno zaplatit cenu odpovídající vzdálenosti k facility. Úkolem je rozhodnout, které facility otevřít a kam připojit klienty. Matematicky vyjádřeno, chceme minimalizovat celkovou cenu shlukování $Q=\sum_{j\in F}(fc)+\sum_{i\in C}(c_{ij})$, kde *fc* je cena za otevření facility a c_{ij} je vzdálenost klienta *i* k facility *j*.

V praxi se množiny facility a klientů často ztotožňují (tj. každý bod je potenciálním středem shluku). Na rozdíl od algoritmu K-means, není nutné předem určit počet shluků. Je potřeba stanovit cenu otevření facility. Ta určuje, jak bude shlukování vypadat. Vysoká hodnota povede ke shlukování s malým počtem velkých shluků; naproti tomu malá hodnota vytvoří mnoho malých shluků.

Popis algoritmu

Nejdříve se vygeneruje přibližné počáteční řešení. To je pak opakovaně vylepšováno lokálními úpravami.

Počáteční řešení se najde následovně. Body jsou zpracovávány v náhodném pořadí. V prvním z nich je vždy otevřena facility. Pro každý další bod je změřena vzdálenost *d* k nejbližší facility. S pravděpodobností *d/fc* (nebo 1, pokud *d* > *fc*) je v bodě otevřena nová facility; jinak je bod přiřazen k nejbližší již otevřené facility.

Následuje iterační proces úprav počátečního řešení. Jedno lokální vylepšení vypadá následovně. Je náhodně vybrána jedna facility a je rozhodnuto, zda by její otevření mohlo vylepšit současné řešení. Předně bude třeba zaplatit cenu za otevření facility. Pro některé body pak bude nová facility blíže než ta stávající. Přeřazením takových bodů k nové facility ušetříme cenu za připojení. Po tomto kroku mohou mít některé facility připojeno jen několik málo bodů. Takové facility může být vhodné uzavřít a body k nim připojené přeřadit jinam. Za připojení bodů jinam sice zaplatíme více, ale uzavřením facility zase mnoho ušetříme.

Celé výše popsané rozhodování provádí funkce, která spočítá, kolik můžeme otevřením vybrané facility ušetřit. Je-li číslo kladné, operace se vyplatí. Facility je otevřena, někteří klienti jsou přeřazeni a některé jiné facility uzavřeny. Řešení je tím vylepšeno. Je-li číslo záporné, zkusíme vybrat jinou facility.

Bylo dokázáno, že lokální vylepšení se mají opakovat celkem n**log(n)* krát, kde *n* je počet bodů. V praxi je možné použít i nižší počet iterací, např. *0,1*n*. Více podrobností se dozvíme v [6].

3.4 Urychlování převzaté knihovny

Nejprve bylo nutné najít v knihovně místa, kde výpočet spotřebuje nejvíce času. To se provede jednoduchou profilací tak, že se v každé metodě změří délka jejího běhu. Pro tento účel vybavíme knihovnu malou strukturou, která si uchová v paměti název metody a její čas. Poté tyto údaje zapíšeme do souboru. Z výsledků těchto experimentů vyplynula snaha urychlit metody *ComputeClustering()*, - pomalý výpočet shlukování, *UpdateVertices()* - rychlý výpočet shlukovaní a nakonec i metodu *GenerateInitialSolution()* - generování počátečního řešení.

Snížení počtu iterací

V metodě *ComputeClustering()* a i *UpdateVertices()* dochází k mnoha voláním metod: *Gain()* tj. vrátí zisk pro daný bod, *Reassign()* tj. provádí přesuny a zavírání facility. Naším cílem bude tedy pokusit se snížit počet volání těchto metod. Původní počet iterací je *0,1*n*,

1 facility se do češtiny přeloží jako příslušenství, což podle mého názoru není vhodný pojem, proto jsem se rozhodl facility nepřekládat.

kde *n* je počet bodů obrázku u metody *ComputeClustering()* a u *Updatevertices() 0,01*n*. Snažili jsme se počet iterací snížit. Zkoušeli jsme lineární, odmocninu a logaritmus.

Zvolme tedy náhodně číslo z intervalu (0,03-0,06), podle kapitoly 4.4 např. 0,04. A podle toho čísla zvolme počet iterací takto *0,04*n* u metody *ComputeClustering()* a počet iterací u metody *UpdateVertices()* ponechme beze změn.

Domnívali jsme se, že pro počet bodů obrázku přesahující 35000 již nebude potřeba takového počtu iterací. Proto jsou konstanty iteračních předpisů voleny tak trochu „podivně“. Do 35 tisíc bodů je shlukování prováděno pečlivěji než při původním řešení, ale pro malé obrázky tento čas navíc nevadí.

Použijme tedy jako předpis pro metodu ComputeClustering() odmocninu tj. $\sqrt{(7n)}+3000$ a metodu *UpdateVertices()* nastavme na $\sqrt{(5n)}+1000$. Takto volený počet iterací měl umožnit shlukovat, jak malé tak i velké obrázky. Nastavit počet iterací pouze jako $\sqrt{n}$, by bylo příliš málo viz kapitola 4.2 měření odchylek.

A nakonec použijeme logaritmický předpis: $1000*\log_2(n)+2000$ pro metodu *ComputeClustering()* a pro metodu *UpdateVertices()* $1000*\log_2(n)+1000$. Nicméně po takovémto dramatickém snížení počtu iterací bylo potřeba zjistit, o kolik jsou výpočty nepřesnější oproti původnímu počtu iterací. Do knihovny přidáme metodu *ComputeError()* pro výpočet odchylky bodů.

Metoda funguje tak, že vezmeme bod a spočítáme vzdálenost *v_b* ke shluku, kam je bod přiřazen. Pak spočítáme vzdálenost *v_n* k nejbližšímu shluku. Zmíněná "odchylka" je pak *v_b-v_n*. Odchylky pak sečteme a vydělíme počtem všech bodů. Výsledný průměr se ještě vydělí *facilitycost* tj. cenou za otevření jednoho shluku (facility), takže by měl být nezávislý na velikosti obrázku. Metoda ještě vrací počet bodů, které mají nenulovou odchylku, tj. těch, které nejsou přiřazené do nejbližšího shluku.

Změny výpočtu počátečního řešení

Urychlování knihovny povedeme pomocí změn generování počátečních řešení. V původní verzi se v metodě *GetClosestFacility()* vrátí vzdálenost nejbližšího shluku k danému bodu. To znamená pro každý bod projít všechny shluky a k tomuto bodu najít nejbližší shluk. To může být časově náročné. Proto zkusíme určitá vylepšení.

V metodě *GetClosestFacility()* nebudeme počítat vzdálenost bodu ke všem shlukům, ale pouze ke 20% všech shluků. Tyto shluky vybereme náhodně. Z takto náhodně vybraných shluků vybereme nejbližší shluk, do kterého bod přiřadíme.

Protože je bod obrázku podobný svému okolí, můžeme se pokusit o další možnost urychlení počátečního řešení použitím mřížky. Pro tento účel jsme si do knihovny přidali informace o velikosti obrázku. Pak ještě přidáme informaci o velikosti jedné buňky, tzv. interval, v základu je nastaven na 10 pixelů. Vytvoříme si matici shluků o rozměrech [(x/interval)+1,(y/interval)+1]. Princip je jednoduchý a velmi rychlý. Bod, který padne do buňky jako první, se stane facility. Ostatní body se pak k tomuto shluku připojí, aniž bychom pak zdlouhavě hledali nejbližší shluk. Nejedná se o tvorbu shluků, je to spíše nástřel řešení, které pak správně vyřeší metoda *ComputeClustering()*.

Urychlování pomocí změn metrik

Zkusme pro urychlení výpočtů použití jiných metrik. Tím se nám výpočty zjednoduší a běh našeho výpočtu by mohl být rychlejší.

Jako první metriku použije manhattanskou metriku, která je na množině $\mathbb{R}^n$ definovaná následujícím vztahem 3.4.1 [7]:

$$\rho(x,y)=|x_1-y_1|+|x_2-y_2|+...+|x_n-y_n| \qquad \textbf{(3.4.1)}$$

zde:

$\rho(x,y)$	*vzdálenost v manhattanské metrice.*
$(x_1, x_2\ldots', x_n)$	*souřadnice bodu x*
$(y_1, y_2\ldots, y_n)$	*souřadnice bodu y*

Jak tedy nová metrika změní naši kombinaci vzdálenosti a barvy dle vztahu 3.2? Doplňme si ji tam. Kombinace jasu a vzdálenosti se změní podle vztahu 3.4.2:

$$d=dx+dy+vahaJasu*d_{rgb} \qquad \textbf{(3.4.2)}$$

kde:

d	*výsledná vzdálenost*
dx	*rozdíl v y-ové souřadnici tj dx=*$\lvert x_1\text{-}y_1\rvert$
dy	*rozdíl v y-ové souřadnici tj dy=*$\lvert x_2\text{-}y_2\rvert$
vahaJasu	*koeficient váhy barvy.*
d_{rgb}	*celkový rozdíl v barvě.*

Dále si ukážeme, jak vypadá klasická euklidovská metrika. Je definována na množině $\mathbb{R}^n$ vztahem 3.4.3 [7]:

$$\rho(x,y)=\sqrt{(x_1-y_1)^2+(x_2-y_2)^2+...+(x_n-y_n)^2} \qquad \textbf{(3.4.3)}$$

A nyní přejděme k euklidovské metrice bez odmocniny, kterou ukazuje vztah 3.4.4.

$$\rho(x,y)=(x_1-y_1)^2+(x_2-y_2)^2+...+(x_n-y_n)^2 \qquad \textbf{(3.4.4)}$$

kde:

$\rho(x,y)$	*vzdálenost v euklidovské metrice bez odmocniny.*
$(x_1-y_1)^2$	*vzdálenost v ose 1.*
$\vdots$	
$(x_n-y_n)^2$	*vzdálenost v ose n.*

Jak tedy další metrika změní naši kombinaci vzdálenosti a barvy, viz vztah 3.2? Doplňme si ji tam. Kombinace jasu a vzdálenosti se změní tak, jak ukazuje vztah 3.4.5.

$$d=dx^2+dy^2+vahaJasu^2*d_{rgb}^2 \qquad \textbf{(3.4.5)}$$

kde:

d	*výsledná vzdálenost*
dx	*rozdíl v y-ové souřadnici tj. dx=*$x_1\text{-}y_1$
dy	*rozdíl v y-ové souřadnici tj. dy=*$x_2\text{-}y_2$
vahaJasu	*koeficient váhy barvy.*
d_{rgb}	*celkový rozdíl v barvě.*

3.5 Spojování podobných shluků a tvorba masky

V této části kapitoly se budeme věnovat spojováním podobných shluků, které mohou najít praktické využití při oddělování objektů nebo v podobě „masky“, která může sloužit k dalším účelům (rekonstrukce obrazu).

Nejprve si připravíme pomocnou strukturu soused, která bude obsahovat informace o sousedech každého shluku a další užitečné informace, např. barvu a číslo shluku (viz Obr. 3.5). Ještě než začneme samotné spojování shluků, je dobré vytvořit si dvourozměrné pole o velikosti obrázku, kam uložíme hranice shluků třeba typu boolean. Hranice shluků najdeme tak, že si vezmeme nějaký pixel a jeho 4 sousedy (vpravo, vlevo, nahoru a dolu) a zjistíme, zda-li je zde rozdíl v „barevné složce“ (v červené, zelené a modré). Pokud ano, pixel zapíši do matice jako „je tu rozdíl“ (true), jinak „není tu rozdíl“ (false). Podle této hranice vyhledáme příslušný shluk, který budeme potřebovat dále při jejich spojování. A teď již ke zmíněnému spojování shluků.

```
public struct Soused
{
    public int cluster;              //index středu shluku
    public Color barva;              //barva shluku
    public int plocha;               //plocha shluku
    public List<int> sousedi;        //sousedi shluku
}
```

Obr. 3.5: *Struktura Soused*

Předpokládejme, že shluky *A* a *B* se mají spojit. To, jestli se mají spojit, nám řekne metoda *vzdalenost()*, která vrací maximu z barevného rozdílu dvou barev po složkách, tj. vrátí maximum ze tří složek. Ještě před samotným spojováním vypočteme celkovou plochu shluků (i s polochou jeho sousedů), kterou budeme potřebovat pro výpočet průměrné barvy. Nyní si spočítáme průměrnou barvu celého spojení těchto dvou shluků jako *novouBarvu*. Touto *novouBarvou* obarvíme shluky *A* a *B*. Dále si připravíme skupiny shluků (jedná se pole s indexy středů shluků), které se budou spojovat. To znamená *skupinu A*, která obsahuje shluk *A* a všechny jeho sousední shluky, a totéž pro *skupinu B*. Nejdříve připojíme takto připravené skupiny do sousedů shluků *A*, *B* a nezapomene těmto sousedům nastavit *novouBarvu*. Při takovémto postupu musíme dávat pozor na duplicitní sousedy. A na závěr připojíme skupiny i do samotných shluků *A* a *B*.

Na závěr vytvoříme masku obrázku tak, že bitmapu obarvíme podle struktury soused. Opět hranice shluků najdeme tak, že si vezmeme nějaký pixel a jeho 4 sousedy a zjistíme, zda-li je zde rozdíl v „barvě“. Pokud ano, pixel zapíšeme jako bílý. Pokud ne, pixel masky bude černý.

4. Experimenty a výsledky

Ještě před závěrečnou sumarizací výsledků, provedeme experimenty a pozorování, ze kterých budeme moci později lépe zhodnotit naše dosažené výsledky.

Parametry hadwarového i softwarového vybavení počítače, na kterém byla měření prováděna, jsou uvedena v Tab. 5

Počítač	CPU: AMD X2 4000+ 2,1Ghz RAM: 6 GB
Systém	Kubuntu 8.04 Heardly Heron x86_64 GNU/Linux 2.6.24-23-generic
Vývojové nástroje	VirtualBox 2.0.4 Microsoft Windows XP Professional se Service Pack 2 Microsoft Visual C# 2005 Express Edition

Tab. 4: *Tabulka s parametry HW a SW vybavení měřícího počítače.*

Popišme si ještě před měřením, pro jaké obrázky budeme všechna měření provádět. Jedná se o tři fotografie, které použijeme vícekrát, přesněji 2 z nich 2x zmenšíme, abychom měli rovnoměrné rozložení velikostí obrázku do 1 mil. pixelů.

Jako první obrázek jsme požili západ slunce, který se skládá z popředí (odlesky vln na mořské hladině), pozadí je tvořeno souvislým pásem pohoří. A nakonec obloha je tvořena mraky s přechodem od žluté barvy k fialové viz příloha č. 2. Jedná se o typickou fotografii, která je vhodná k oddělení pozadí, mraku atd.

Jako druhý uvedeme fotografii ZČU v létě. Popředí je tvořeno řepkou, která vytváří drobné detaily v různých barvách (červená,zelená, oranžová). Na pozadí je vidět ZČU, která je obklopena tmavě zelenými stromy. Obloha je tvořena mraky viz příloha č. 2 . Fotografie je vhodná nejenom pro drobné plody řepky, ale také pro větší plochy stromů.

Jako poslední experimentální obrázek použijeme fotografii trávy, která se skládá z dlouhých listů, po dešti viz příloha č. 2. Obrázek je zajímavý z hlediska výskytu dlouhých listů vhodných k segmentaci.

V částech této kapitoly 4.7, 4.8, 4.9 a 4.10 budeme vždy používat dvojice obrázků. K barevným obrázkům přidáme příslušný šedotónový, barvy v něm jsou voleny náhodně pro lepší identifikaci shluků. *Pb* se u obrázků v kapitole 4.8 pohybuje okolo 10% a v kapitole 4.9 3%.

Ještě si ukažme smysl parametrů „Váha jasu“ (viz vztah 3.2) a „Velikost shluků“ (viz kapitola 3.3, jedná se o cenu otevření jedné facility). Parametr „Vážení jasu“ (váhaJasu) ovlivňuje citlivost změny v barvě, tj. při malé citlivosti se projeví jen velká změna. Naopak při velkém důrazu na citlivost v barvě se projeví i malé odchylky v barvě. Parametr „Velikost shluků“ mění velikost shluků, čím vyšší hodnota, tím jsou shluky „větší“. Větší shluky umožňují větší kompresi, menší zachovávají lepší kvalitu.

Přidejme ještě vysvětlení použitých zkratek:

Pb – Poměr počtu bodů s odchylkou, tj. těch které nejsou přiřazené do nejbližšího shluku vůči všem bodům obrázku.

Ob – Průměrná odchylka bodů (součet rozdílů vzdáleností: bodu od nejbližšího shluku a od shluku kam je bod přiřazen).

4. 1 Snižování počtu iterací

Naše první 4 měření se týkají snížení časové náročnosti programu, snížením počtu iterací. Parametr, který jsme měnili, byl počet iterací, které provádí metoda *Computeclustering()* a metoda *Updatevertices()*. Výsledky experimentů jsou k dispozici též v obrazové podobě na přiloženém CD.

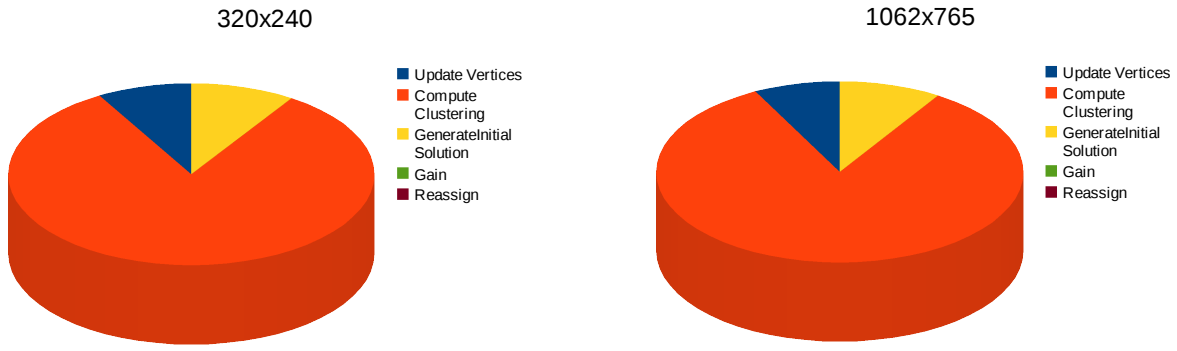
První měření mělo za úkol zjistit, kde program stráví nejvíce času, tzn. určit, která metoda výpočet brzdí. To se povedlo, jak ukazují tabulky 4.1.1 a 4.1.2 a grafů 4.1.1 a 4.1.1 Z tabulek vidíme, že při zvyšovaní počtu bodů obrázku extrémně narůstá především výpočetní doba metody *ComputeClustering()*, a že se podílí na celkovém výpočtu asi z 80%. Z grafů neplyne lineární závislost ačkoliv používáme počet iterací zvolený *0,1*n*. To je způsobeno metodami *Gain()* a *Reassign()*, ve kterých se také pracuje se všemi body obrázku.

	Velikost obrázku v pixelech						
	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
	76800	172800	480000	378573	649800	812430	1024000
Název metody	**Časy jednotlivých metod v sekundách**						
Update Vertices	13,13	57,15	450,66	279,64	808,12	1374,95	2014,04
Compute Clustering	129,8	668,37	5189,77	3172,27	9392,04	15061,35	22591,39
GenerateInitial Solution	14,54	107,58	461,17	248,58	1085,94	1619,48	3175,98
Gain	0,01	0,03	0,09	0,08	0,11	0,16	0,2
Reassign	0	0	0	0	0,01	0	0
Celkový čas výpočtu [s]	157,5	833,16	6101,81	3700,67	11286,32	18056,05	27781,78

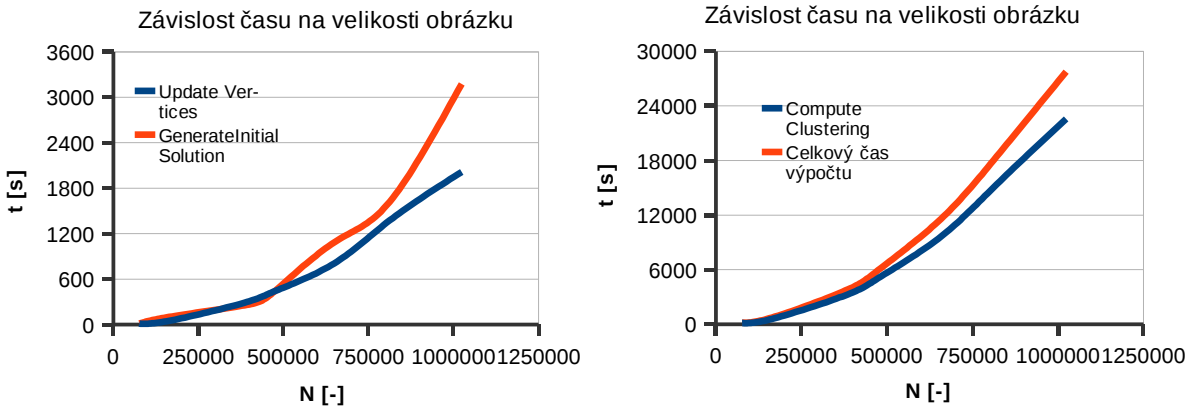
Tab. 4.1.1: *Naměřené časy jednotlivých metod v sekundách*

	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Procenta jednotlivých částí výpočtu**						
Update Vertices	0,08	0,07	0,07	0,08	0,07	0,08	0,07
Compute Clustering	0,82	0,8	0,85	0,86	0,83	0,83	0,81
GenerateInitial Solution	0,09	0,13	0,08	0,07	0,1	0,09	0,11
Gain	0	0	0	0	0	0	0
Reassign	0	0	0	0	0	0	0
Celkový čas výpočtu [%]	1	1	1	1	1	1	1

Tab. 4.1.1: *Naměřené časy jednotlivých metod v procentech*



Graf 4.1.1: *Naměřené časy jednotlivých metod v procentech. Zde je vidět, že největší procento času zabírá metoda ComputeClustering().*



Graf 4.1.2: *Naměřené časy jednotlivých metod v sekundách. Zde je vidět, že největší výpočetní čas spotřebuje metoda ComputeClustering().*

Druhé měření mělo za úkol snížení počtu iterací u metody *ComputeClustering()* pomocí lineárního vztahu $počet\ iterací = 0{,}04 * n$, kde n je počet bodů obrázku. Metodu *UpdateVertices()* ponecháme beze změn.

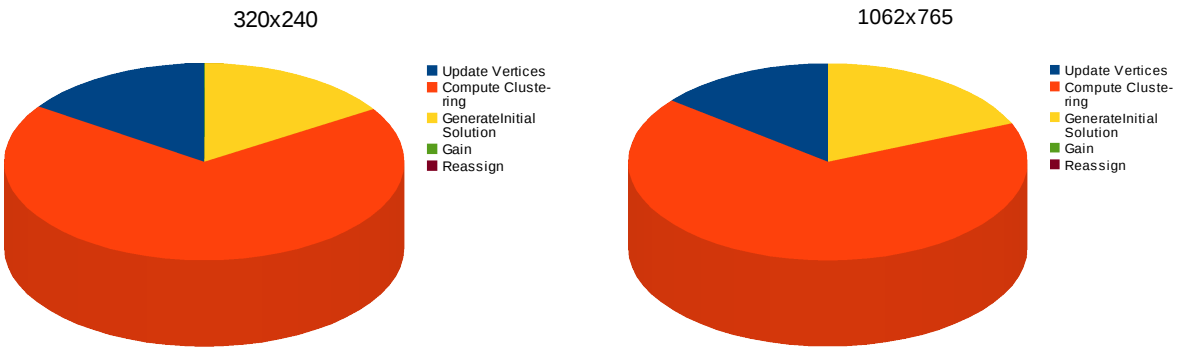
Z tabulky 4.1.3 je vidět, že metoda *ComputeClustering()* se zrychlila zhruba o ½ oproti původnímu řešení s počtem iterací *0,1*n*. Z tabulky 4.1.4, že zde metoda *ComputeClustering()* tráví přibližně o 15% méně času z celkové doby výpočtu. Z grafu 4.1.3 je vidět mírné přerozdělení procent výpočtu dle Tab. 4.1.4. Dále pak graf 4.1.4 ukazuje, že se typ křivek grafu 4.1.2 vůbec nezměnil, ale pouze se snížila výpočetní doba metody C*omputeclustering()* a tím i doba celkového výpočtu.

	Velikost obrázku v pixelech						
	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
	76800	172800	480000	378573	649800	812430	1024000
Název metody	**Časy jednotlivých metod v sekundách**						
Update Vertices	11,17	53,75	419,62	257,07	747,54	1177,33	1843,9
Compute Clustering	49,24	245,45	1921,66	1187,13	3451,54	5496,19	8581,11
GenerateInitial Solution	11,49	54,85	486,66	453,97	1049,75	1530,71	2869,21
Gain	0,02	0,03	0,1	0,07	0,12	0,15	0,14
Reassign	0,01	0	0,01	0	0	0,01	0
Celkový čas výpočtu [s]	71,95	354,12	2828,12	1898,32	5249,09	8204,51	13294,62

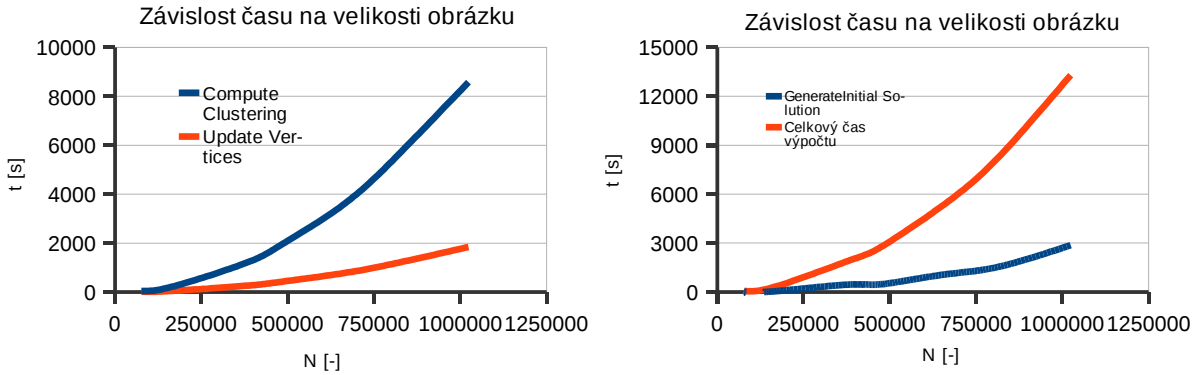
Tab. 4.1.3: *Naměřené časy jednotlivých metod v sekundách.*

	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Procenta jednotlivých částí výpočtu**						
Update Vertices	0,16	0,15	0,15	0,14	0,14	0,14	0,14
Compute Clustering	0,68	0,69	0,68	0,63	0,66	0,67	0,65
GenerateInitial Solution	0,16	0,15	0,17	0,24	0,2	0,19	0,22
Gain	0	0	0	0	0	0	0
Reassign	0	0	0	0	0	0	0
Celkový čas výpočtu [%]	1	1	1	1	1	1	1

Tab. 4.1.4: *Naměřené časy jednotlivých metod v procentech.*



Graf 4.1.3: *Naměřené časy jednotlivých metod v procentech. Zde je vidět, že největší procento času zabírá metoda ComputeClustering().*



Graf 4.1.4: *Naměřené časy jednotlivých metod v sekundách. Je tu vidět jisté zlepšení, ale typ polynomiální závislosti se vůbec nezměnil. Stále zůstala vysoká výpočetní náročnost u metody ComputeClustering().*

Třetí měření mělo za úkol výrazně snížit počet iterací u metody *ComputeClustering()* a *UpdateVertices()* a zajistit tak v rozumném čase zpracování velkých obrázků. Tedy u metody *ComputeClustering()* byl stanoven počet iterací vztahem $počet\ iterací = \sqrt{7 * n} + 3000$, kde n je počet bodů obrázku. U metody *Updatevertices()* vztahem $počet\ iterací = \sqrt{5 * n} + 1000$.

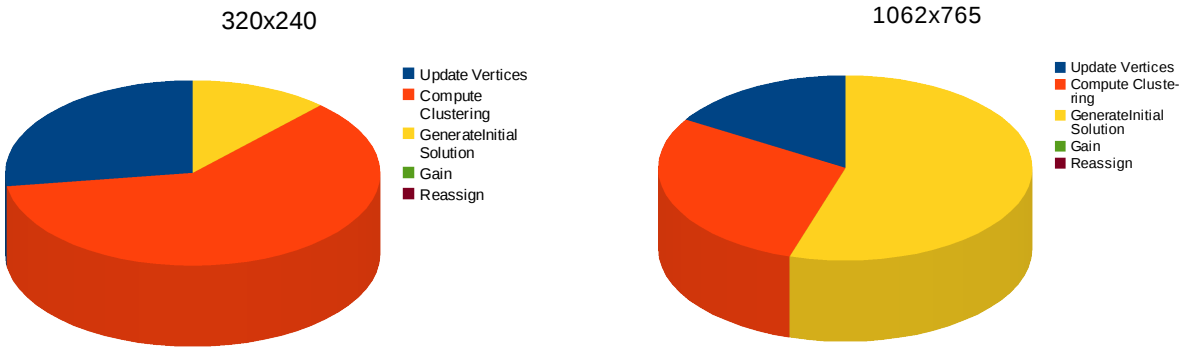
Z tabulky 4.1.5 je vidět, že metoda *ComputeClustering()* se zrychlila zhruba 10x oproti původní verzi a metoda *UpdateVertices()* je přibližně o 50% rychlejší pro velké obrázky, viz tabulky 4.1.1 a 4.1.2. Dále z tabulky 4.1.6 je vidět, že zde metoda *ComputeClustering()* již netráví z hlediska celkového výpočtu tolik času pro velké obrázky. Z grafu 4.1.5 je vidět výrazné přerozdělení procent výpočtu dle Tab. 4.1.6. Celkovou dobu výpočtu již neudává metoda *ComputeClustering()*, ale generování počátečního řešení. To je způsobeno výrazným snížením počtu iterací této metody, z toho plyne potřeba změn výpočtů počátečního řešení viz 4.5. Dále pak graf 4.1.6 ukazuje, že se typ křivek grafu 4.1.2 velmi výrazně přiblížil k lineární závislosti.

	Velikost obrázku v pixelech						
	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
	76800	172800	480000	378573	649800	812430	1024000
Název metody	**Časy jednotlivých metod v sekundách**						
Update Vertices	28,15	72,51	266,22	184,9	373,52	504,28	732,13
Compute Clustering	62,56	156,3	513,17	363,21	701,25	896,51	1300,36
GenerateInitial Solution	12,26	56,79	480,73	437,06	1192,4	1698,67	3238,52
Gain	0,01	0,05	0,09	0,09	0,15	0,19	0,19
Reassign	0	0	0	0	0	0	0
Celkový čas výpočtu [s]	103,01	285,66	1260,27	985,37	2267,46	3099,81	5271,37

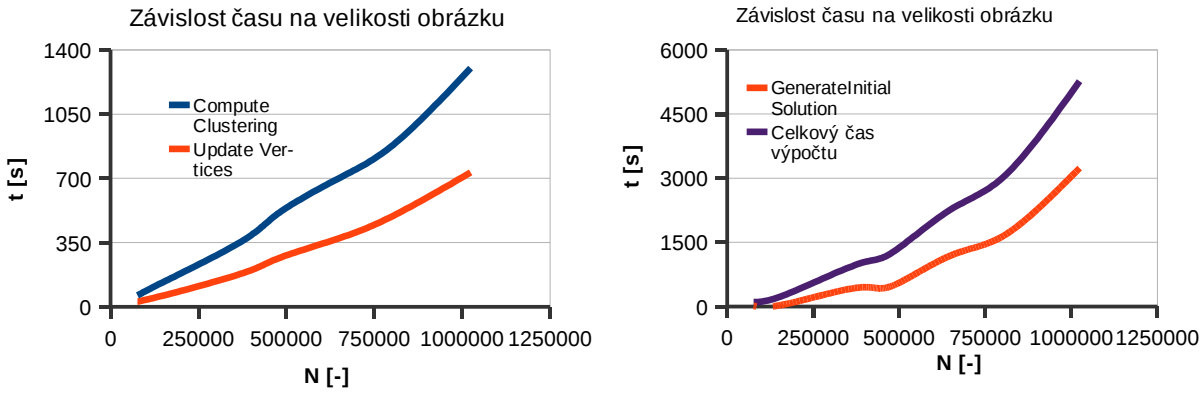
Tab. 4.1.5: *Naměřené časy jednotlivých metod v sekundách, zde je vidět velmi výrazné urychlení výpočtu.*

	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Procenta jednotlivých částí výpočtu**						
Update Vertices	0,27	0,25	0,21	0,19	0,16	0,16	0,14
Compute Clustering	0,61	0,55	0,41	0,37	0,31	0,29	0,25
GenerateInitial Solution	0,12	0,2	0,38	0,44	0,53	0,55	0,61
Gain	0	0	0	0	0	0	0
Reassign	0	0	0	0	0	0	0
Celkový čas výpočtu [s]	1	1	1	1	1	1	1

Tab. 4.1.6: *Naměřené časy jednotlivých metod v procentech ukazují, že většinu času výpočtu již tvoří metoda GenerateInitialSolution() pro větší obrázky.*



Graf 4.1.5: *Naměřené časy jednotlivých metod v procentech. Zde je vidět změna poměru výpočtu metody ComputeClustering() a metody GenerateInitialSolution(). Celková výpočetní doba se přesouvá díky snížení počtu iterací na generování počátečního řešení.*



Graf 4.1.6: *Naměřené časy jednotlivých metod v sekundách. Je zde vidět výrazné zlepšení, a změna typu závislosti u metody ComputeClustering(). Je vidět, že čas výpočtu již brzdí metoda GenerateInitialSolution().*

Čtvrté a poslední měření mělo za cíl ještě více zrychlit metody *ComputeClustering()* a *UpdateVertices()*. Toto se podařilo, ale takto již drasticky snížený počet iterací pomalu začínal mít vliv na odchylky bodů viz kapitola 5.3 Měření odchylek. Stanovení počtu iterací bylo provedeno takto: u metody *ComputeClustering()* $počet\,iterací = 100 * \log_2(n) + 2000$, kde n je počet bodů obrázku a u metody *UpdateVertices()* *počet iterací je* $100 * \log_2(n) + 1000$.

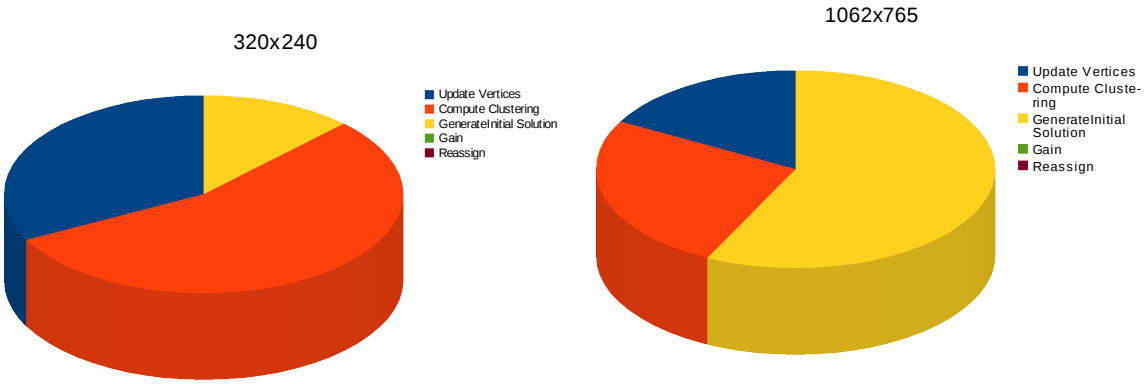
Z tabulek 4.1.7, 4.1.8 a grafu 4.1.7, 4.1.8 je vidět, že metody *ComputeClustering()* *UpdateVertices()* již nejsou tak pomalé jako tomu bylo u lineárního počtu iterací viz tabulky 4.1.1, 4.1.3. Naopak je vidět, že další urychlení je potřeba provést v metodě *GeneralinitialSolution()* viz kapitola 4.5.

	Velikost obrázku v pixelech						
	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
	76800	172800	480000	378573	649800	812430	1024000
Název metody	**Časy jednotlivých metod v milisekundách**						
Update Vertices	36,04	90,18	259,27	194,48	347,98	439,65	538,56
Compute Clustering	60,71	138,4	396,7	300,83	531,54	662,76	858,04
GenerateInitial Solution	13,64	57,93	539,42	217,86	1082,24	1478,48	3037,37
Gain	0,01	0,03	0,08	0,07	0,14	0,15	0,16
Reassign	0	0	0	0	0	0,02	0
Celkový čas výpočtu [s]	110,4	286,57	1195,53	713,32	1961,99	2581,25	4434,37

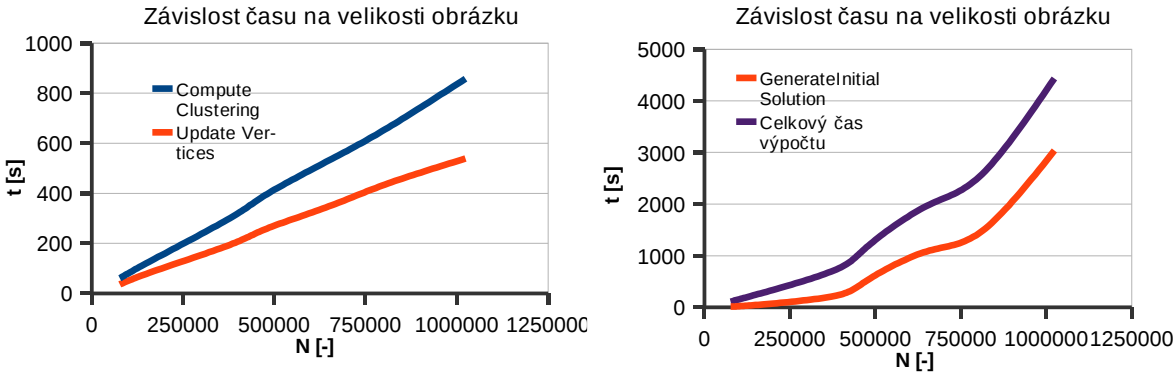
Tab. 4.1.7: *Naměřené časy jednotlivých metod v sekundách. Vidíme zde potřebu urychlit metodu GenerateInitialSolution().*

	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Procenta jednotlivých částí výpočtu**						
Update Vertices	0,33	0,31	0,22	0,27	0,18	0,17	0,12
Compute Clustering	0,55	0,48	0,33	0,42	0,27	0,26	0,19
GenerateInitial Solution	0,12	0,2	0,45	0,31	0,55	0,57	0,68
Gain	0	0	0	0	0	0	0
Reassign	0	0	0	0	0	0	0
Celkový čas výpočtu [s]	1	1	1	1	1	1	1

Tab 4.1.8: *Naměřené časy jednotlivých metod v procentech ukazují, že většinu času výpočtu, pro větší obrázky tvoří metoda GenerateInitialSolution(), proto je potřeba ji urychlit viz kapitola 4.4*



Graf 4.1.7: *Naměřené časy jednotlivých metod v procentech. Jsou tu vidět výrazné změny poměrů jednotlivých metod výpočtu ,především ComputeClustering(), dále metody GenerateInitialSolution() a UpdateVertices() .*



Graf 4.1.8: *Naměřené časy jednotlivých metod v sekundách. Grafy metod ComputeClustering() a UpdateVertices() tvoří již lineární závislost na čase. Naopak celkovou dobu výpočtu, jak je vidět z grafu, již udává GenerateInitialSolution().*

A nyní proveďme menší souhrn našich čtyř měření v závěrečné tabulce 4.1.9. Z tabulky je vidět postupné urychlení celého programu pomocí snížení počtu iterací metod *ComputeClustering()* a *UpdateVertices()*, především pro velké obrázky, kde původní verze je téměř nepoužitelná.

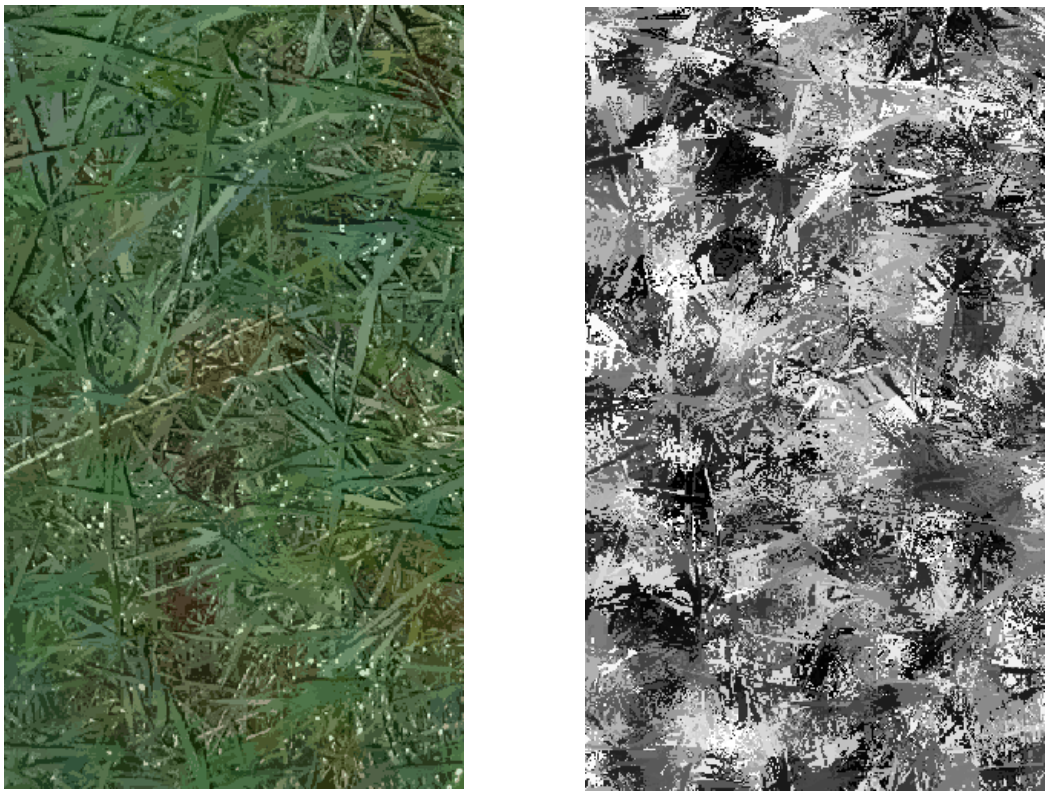
Dále si ukažme, jaký vliv mají tyto změny počtu iterací na výsledné obrázky. Použijeme metodu *ComputeClustering()* s nastavením s parametry (*Váha jasu* = 1 a *velikost shluků* = 1), tj. základní nastavení programu po spuštění. K barevnému obrázku (a) vždy přiložíme černobílý obrázek (b) , abychom lépe viděli hranice jednotlivých shluků.

Obrázek 4.1.1 ukazuje původní pomalé řešení s počtem iterací = *0,1*n*, dále pak obrázek 4.1.2 ukazuje o něco rychlejší řešení s počtem iterací = *0,04*n*. Předposlední obrázek 4.1.3 zobrazuje řešení s *počtem iterací* $=\sqrt{(7n)}+3000$. A nakonec poslední obrázek 4.1.4 ukazuje řešení s *počtem iterací* $=100*\log_2(n)+2000$. Pouhým okem se jen těžko hledá „viditelný rozdíl“ mezi obrázky. Jak moc jsou nepřesné se dozvíme v kapitole 4.2. Jedná se vždy o poslední obrázek, tj. o ten s největší velikostí. Pokusme se ještě jako míru kvality obrazu použít výpočet PSNR. Měření PSNR provedeme vůči Obr. 4.1.1 (barevná verze) s obrázky 4.1.2, 4.1.3 a 4.1.4.

Název metody	**Velikost obrázku v pixelech**						
	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
	76800	172800	480000	378573	649800	812430	1024000
Původní řešení							
Update Vertices	13,13	57,15	450,66	279,64	808,12	1374,95	2014,04
Compute Clustering	129,8	668,37	5189,77	3172,27	9392,04	15061,35	22591,39
Celkový čas výpočtu [s]	**157,5**	**833,16**	**6101,81**	**3700,67**	**11286,32**	**18056,05**	**27781,78**
Druhý experiment							
Update Vertices	11,17	53,75	419,62	257,07	747,54	1177,33	1843,9
Compute Clustering	49,24	245,45	1921,66	1187,13	3451,54	5496,19	8581,11
Celkový čas výpočtu [s]	**71,95**	**354,12**	**2828,12**	**1898,32**	**5249,09**	**8204,51**	**13294,62**
Třetí experiment							
Update Vertices	28,15	72,51	266,22	184,9	373,52	504,28	732,13
Compute Clustering	62,56	156,3	513,17	363,21	701,25	896,51	1300,36
Celkový čas výpočtu [s]	**103,01**	**285,66**	**1260,27**	**985,37**	**2267,46**	**3099,81**	**5271,37**
Čtvrtý experiment							
Update Vertices	36,04	90,18	259,27	194,48	347,98	439,65	538,56
Compute Clustering	60,71	138,4	396,7	300,83	531,54	662,76	858,04
Celkový čas výpočtu [s]	**110,4**	**286,57**	**1195,53**	**713,32**	**1961,99**	**2581,25**	**4434,37**

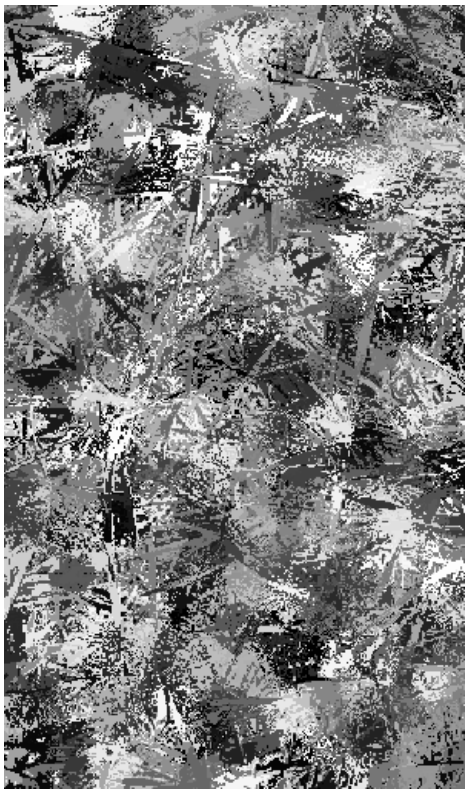
Tab. 4.1.9: *Souhrnná tabulka měření časů. Tabulka ukazuje postupné výrazné urychlení programu.*

Obr. 4.1.1 (a) a (b): *Obrázky ukazují původní řešení s počtem iterací 0,1*n.*



Obr. 4.1.1 (a) a (b): *Obrázky ukazují druhé řešení s počtem iterací 0,04*n. PSNR 14.74 [dB] ukazuje na odlišnost od původního počtu iterací.*

Obr. 4.1.3 (a) a (b): *Obrázky ukazují třetí řešení s počtem iterací* $= \sqrt{(7\mathrm{n})} + 3000$ *. Opět, jak moc je jiný, provedeme pomoci PSNR 14.76 [dB]. Znovu se obrázek liší od původní verze.*



Obr. 4.1.4 (a) a (b): *Obrázky ukazují poslední řešení s počtem iterací:* $100*\log_2(n)+2000$ *. Znovu změříme PSNR, které vychází 14.75 [dB].Opět ukazuje, že se obrázky liší.*

4. 2 Měření odchylek při změně počtu iterací

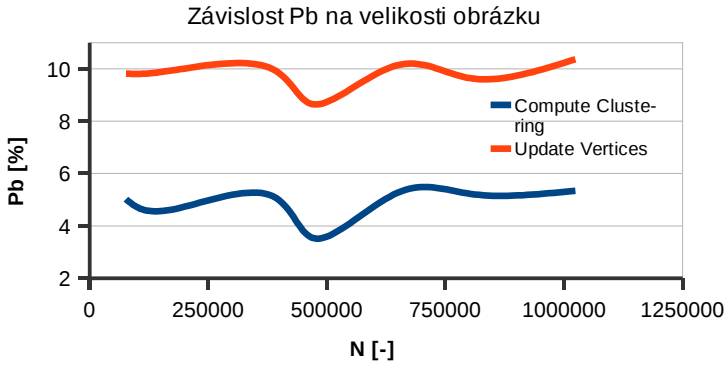
Naše další čtyři měření měla za úkol ukázat, jak moc se při snížení počtu iterací zvýší *Ob* a *Pb*. *Pb* je uveden v procentech.

Jako první provedeme měření základní verze programu bez snížení počtu iterací, tj pro metodu *ComputeClustering()* $počet\ iterací = 0{,}1*n$, kde *n* je počet bodů obrázku, a pro metodu *UpdateVertices()* $počet\ iterací = 0{,}01*n$.

Z tabulky 4.2.1 a grafu 4.2.1 je vidět, že se *Pb* udržuje kolem 5% až na jednu výjimku, která je způsobena pro nás vhodným rozložením bodů obrázku. Totéž platí pro metodu *UpdateVertices()* s tím, že se *Pb* udržuje kolem 10%.

	Velikost obrázku	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Typ Odchylky**							
Compute Clustering	Ob	0,0034	0,0024	0,0008	0,0026	0,0018	0,0016	0,0017
	Pb [%]	5,025	4,619	3,511	5,190	5,272	5,200	5,345
Update Vertices	Ob	0,0029	0,0022	0,0012	0,0021	0,0016	0,0014	0,0015
	Pb [%]	9,816	9,932	8,637	10,044	10,140	9,615	10,363

Tab.4.2.1: *Velikost odchylek. Z tabulky vidíme, že se zvětšující se velikostí obrázku se odchylky příliš nezvyšují.*



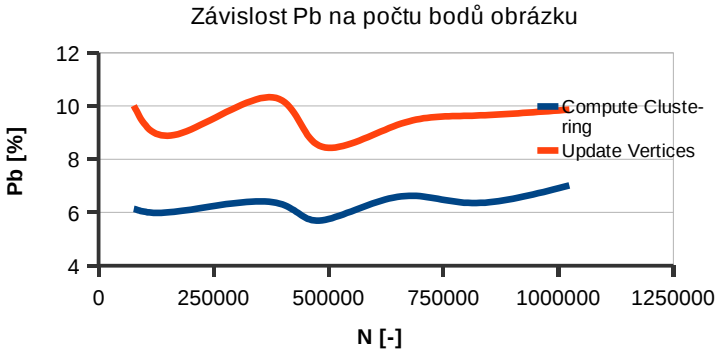
Graf 4.2.1: *Naměřené odchylky metod ComputeClustering() a UpdateVertices().*

Jako druhé provedeme měření snížení počtu iterací u metody *ComputeClustering()* z původního $počtu\,iterací = 0{,}1 * n$ na $počet\,iterací = 0{,}04 * n$, kde n je počet bodů obrázku, a metodu *UpdateVertices()* ponecháme beze změny, tj. $počet\,iterací = 0{,}01 * n$.

Z tabulky 4.2.2 a grafu 4.2.2 je vidět, že se *Pb* mírně zvýšil oproti původní verzi viz Tab. 4.2.1 na přibližně 6-7%. A metoda *UpdateVertices()* zůstala beze změny na přibližně 10%.

	Velikost obrázku	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Typ Odchylky**							
Compute Clustering	Ob	0,0041	0,0030	0,0019	0,0030	0,0023	0,0020	0,0022
	Pb [%]	6,135	6,041	5,695	6,391	6,588	6,356	7,020
Update Vertices	Ob	0,0031	0,0021	0,0014	0,0021	0,0016	0,0014	0,0015
	Pb [%]	10,008	8,943	8,508	10,325	9,287	9,635	9,865

Tab 4.2.2: *Velikost odchylek. Z tabulky vidíme, že se odchylky příliš nezvyšují.*



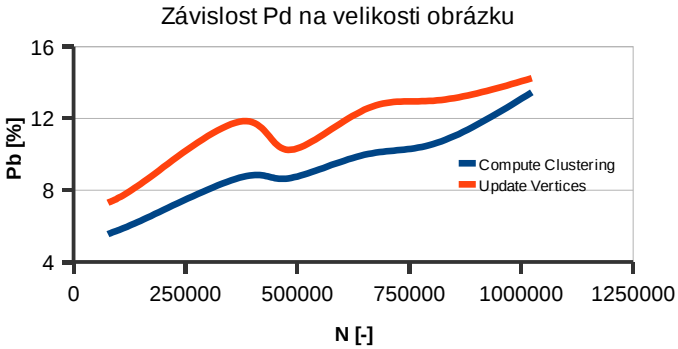
Graf 4.2.2: *Naměřené odchylky metod ComputeClustering() a UpdateVertices(), opět je vidět, že Pb nemá tendenci růstu.*

Jako předposlední provedeme měření snížení počtu iterací u metody *ComputeClustering()* na $počet\,iterací = \sqrt{7 * n} + 3000$, kde n je počet bodů obrázku a metodu *UpdateVertices()* změníme na $počet\,iterací = \sqrt{5 * n} + 1000$.

Z grafu 4.2.3 a tabulky 4.2.3 je vidět, že *Pb* roste a nelze ho tedy odhadnout v mezemi.

	Velikost obrázku	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Typ Odchylky**							
Compute Clustering	Ob	0,0039	0,0035	0,0029	0,0046	0,0035	0,0033	0,0040
	Pb [%]	5,555	6,567	8,658	8,746	9,968	10,640	13,450
Update Vertices	Ob	0,0025	0,0021	0,0017	0,0024	0,0019	0,0018	0,0019
	Pb [%]	7,294	8,762	10,244	11,845	12,481	12,999	14,238

Tab. 4.2.3: *Velikost odchylek. Z tabulky vidíme, že se odchylky začínají zvyšovat.*



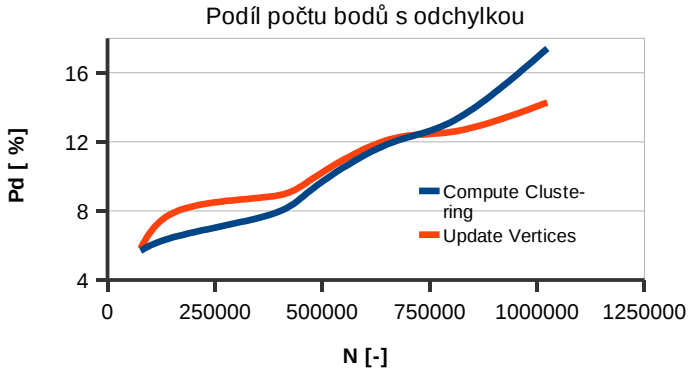
Graf 4.2.3: *Naměřené odchylky metod ComputeClustering() a UpdateVertices(), z grafu je vidět tendence růstu s mírnými výkyvy, které jsou způsobeny uspořádáním pixelů.*

Jako čtvrté a poslední měření jsme provedli snížení počtu iterací u metody *ComputeClustering()* na $počet\,iterací = 100 * \log_2(n) + 2000$ a u metody *UpdateVertices() na* $počet\,iterací = 100 * \log_2(n) + 1000$, kde *n* je počet bodů obrázku.

Z grafu 4.2.4 a tabulky 4.2.4 je vidět, že *Pb* již výrazně narůstá. Na základě měření jsme se rozhodli zavrhnout takové volby počtu iterací.

	Velikost obrázku	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Typ Odchylky**							
Compute Clustering	Ob	0,0040	0,0035	0,0032	0,0024	0,0040	0,0038	0,0046
	Pb [%]	5,693	6,601	9,333	7,784	11,837	13,315	17,404
Update Vertices	Ob	0,0023	0,0020	0,0016	0,0013	0,0019	0,0018	0,0020
	Pb [%]	5,815	8,086	9,911	8,829	12,091	12,623	14,278

Tab. 4.2.4: *Velikost odchylek. Vidíme výrazné zvyšování Pb, avšak absolutní odchylka bodů se příliš nezvyšuje.*



Graf 4.2.4: *Naměřené odchylky metod ComputeClustering() a UpdateVertices(). Z grafu je vidět vysoký nárůst Pb.*

A nyní opět provedeme menší souhrn našich měření. Budou nás zajímat metody *ComputeClustering()* a *UpdateVertices().* Dále pak samozřejmě jejich naměřené *Pb*. Z tabulky 4.2.5 je vidět, že se zmenšujícím se počtem iterací začíná stoupat *Pb* a u logaritmického počtu iterací velice výrazně. Ukazuje se potřeba lineárního počtu iterací, pro dosažení konstantní odchylky viz 4.3. To je nepříjemné zejména pro velké obrázky. Také tím vyvracíme předpoklad z 3.4 (snižování počtu iterací) o menším počtu iterací.

Velikost obrázku	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
Název metody	**Pb v [%]**						
Původní řešení							
Compute Clustering	5,025	4,619	3,511	5,190	5,272	5,200	5,345
Update Vertices	9,816	9,932	8,637	10,044	10,140	9,615	10,363
Druhé řešení							
Compute Clustering	6,135	6,041	5,695	6,391	6,588	6,356	7,020
Update Vertices	10,008	8,943	8,508	10,325	9,287	9,635	9,865
Třetí řešení							
Compute Clustering	5,555	6,567	8,658	8,746	9,968	10,640	13,450
Update Vertices	7,294	8,762	10,244	11,845	12,481	12,999	14,238
Poslední řešení							
Compute Clustering	5,693	6,601	9,333	7,784	11,837	13,315	17,404
Update Vertices	5,815	8,086	9,911	8,829	12,091	12,623	14,278

Tab. 4.2.5: *Tabulka souhrnných výsledků měření Pb. Z tabulky plyne nárůst Pb při snižování počtu iterací.*

4. 3 Měření závislosti počtu iterací při konstantním Pb

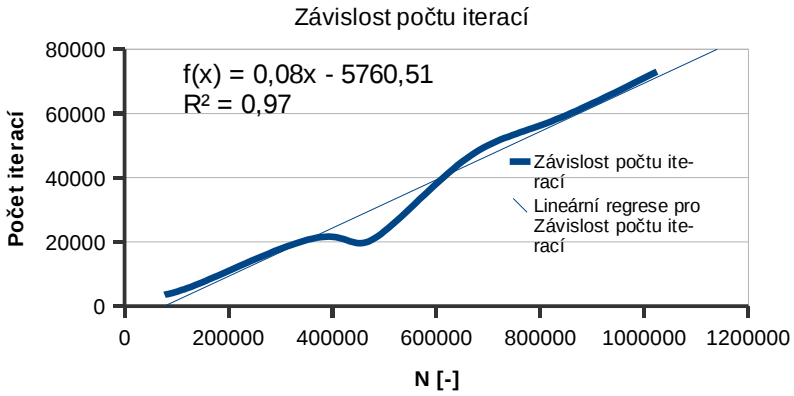
Naše další měření má za úkol stanovit počet iterací potřebný k dosažení *Pb*, sledujeme tedy počet iterací, které je potřeba provést.

Měření bylo prováděno pro konstantu: *Pb = 6%*. Z důvodů časové náročnosti výpočtu pro velké obrázky, jsme zvolili o něco málo vyšší Pb než tu, která byla změřena v původním řešení.

Z tabulky 4.3 a grafu 4.3 je vidět, že pro dosažení Pb je potřeba lineární závislost, tj. *počet iterací = a * n*, *a* je koeficient lineárního předpisu a *n* je počet bodů obrázku.

	Velikost obrázku v pixelech						
	320x240	480x360	800x600	571x663	950x684	1062x765	1280x800
	76800	172800	480000	378573	649800	812430	1024000
Měřený počet iterací	3500	9000	21000	21500	45000	57000	73000
Čas výpočtu [s]	52,7	303,43	1861,93	1634,32	5685,98	7263,53	13419,98

Tab. 4.3: *Závislost počtu iterací na konstantní odchylce 6%. Z tabulky je vidět, že se počet iterací zvyšuje.*



Graf 4.3: *Závislost počtu iterací při konstantní odchylce 6%. Z lineární regrese a koeficientu korelace plyne, že bude potřeba lineárního počtu iterací.*

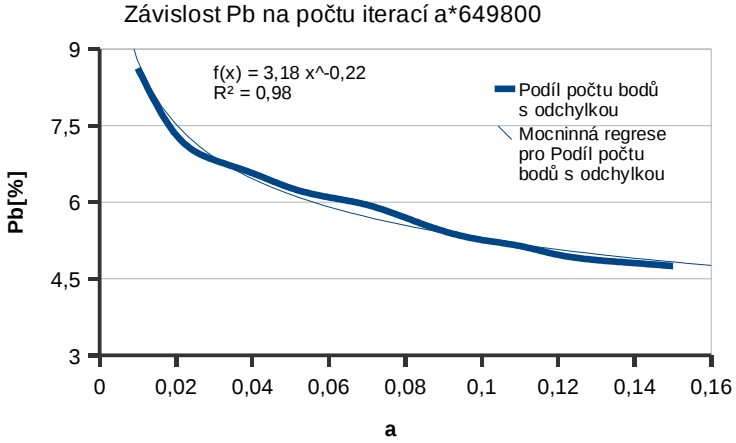
4. 4 Měření závislosti odchylky na počtu iterací

Náš další experiment spočívá ve stanovení *Pb* v závislosti na počtu iterací tj. závislosti a**n*. Parametr, který budeme sledovat je tedy *Pb*. Měření provedeme pro jeden obrázek s počtem bodů *n = 649800*, který vybereme náhodně.

Z výsledku měření, jak ukazuje tabulka 4.4 a graf 4.4, bylo zjištěno, že Pb připomíná hyperbolu, ve skutečnosti je to však průběh mocninné funkce. Z výsledků vyplývá, že vhodný předpis pro počet iterací bude mezi *0,03-0,06 n*, protože *Pb* se zde pohybuje mezi *7-6%*. Dále pak k dosažení takového poměru odchylky potřebujeme konkrétní počet iterací, viz graf 4.4 tj. měníme koeficient *a*, kde *počet iterací = a *n* a *n* je konstantní počet bodů obrázku.

Pb [%]	8,622	7,320	6,826	6,570	6,279	6,097	5,947
Počet iterací a*n	6498	12996	19494	25992	32490	38988	45486
Lin. Předpis a	0,010	0,020	0,030	0,040	0,050	0,060	0,070
Pb [%]	5,697	5,434	5,262	5,142	4,969	4,867	4,809
Počet iterací a*n	51984	58482	64980	71478	77976	84474	90972
Lin. Předpis n	0,080	0,090	0,100	0,110	0,120	0,130	0,140

Tab. 4.4: *Závislosti Pb počtu iterací. Z tabulky plyne snížení Pb.*



Graf 4.4: *Závislosti Pb na lineárním předpis. Z koeficientu korelace plyn mocninná závislost:* $3{,}18 * a^{-0{,}22}$.

4. 5 Urychlování počátečního řešení

Z výsledků třetího a čtvrtého experimentu viz kapitola 4.1 plyne potřeba urychlit generování počátečního řešení. Urychlení provedeme ve dvou experimentech, kde měříme časy v sekundách hlavně metod *GenerateInitialSolution()*, *ComputeClustering()* a odchylky metod *ComputeClustering()* a *UpdateVertices()*. Zvolený počet iterací je stejný jako u třetího měření, viz kapitola 4.1.

První experiment spočívá ve vybrání (20% shluků) náhodně. Viz kapitola 3.4 změny výpočtu počátečního řešení.

Z tabulky 4.5.1 je vidět přibližně trojnásobné urychlení metody *GenerateInitialSolution()* a dále pak žádné zhoršení metody *ComputeClustering ()*, což vede k celkovému výraznému urychlení výpočtu.

Dále pak vidíme z tabulky 4.5.2, že se P*b* výrazně snížil.

	Velikost obrázku v pixelech		
	800x600	950x684	1280x800
Název metody	480000	649800	1024000
UpdateVertices	231,2	351,95	638,45
ComputeClustering	502,9	718,13	1250,38
GenerateInitialSolution	181,57	380,79	1035,19
gain	0,09	0,12	0,16
Reassign	0	0	0
Celkový čas výpočtu [s]	915,83	1451,09	2924,48

Tab. 4.5.1: *Úprava počátečního řešení pomocí náhodného výběru shluků*

		Velikost obrázku v pixelech		
Název metody	**Typ Odchylky**	800x600	950x684	1280x800
Compute Clustering	Ob	0,0026	0,0028	0,0033
	Pb [%]	4,854	4,181	3,399
Update Vertices	Ob	0,0017	0,0018	0,0019
	Pb [%]	10,180	11,329	11,901

Tab. 4.5.2: *Velikost odchylek. Z tabulky je vidět jejich výrazné snížení u metody ComputeClustering() a u metody UpdateVertices() její mírné zlepšení.*

Další experiment spočívá v použití mřížky o velikosti 20 pixelů, viz kapitola 3.4 změny výpočtu počátečního řešení.

Z tabulky 4.5.3 je vidět, že počáteční řešení je okamžitě, tj. metoda *GeneralInitialSolution()*. Dále je zde vidět zhoršení metody *ComputeClustering()*, která musí provádět více přesouvání bodů mezi shluky, protože počáteční řešení je nepřesné, oproti řešení bez urychlení viz kapitola 4.1 třetí měření.

Z tabulky 4.5.4 plyne výrazné snížení *Pb* viz kapitola 4.2 třetí měření.

	Velikost obrázku v pixelech		
	800x600	950x684	1280x800
Název metody	480000	649800	1024000
UpdateVertices	233,53	354,38	639,05
ComputeClustering	730,26	1136,24	2268,78
GenerateInitialSolution	0,38	0,49	0,92
gain	0,08	0,14	0,17
Reassign	0	0	0
Celkový čas výpočtu [s]	964,32	1491,35	2909,07

Tab. 4.5.3: *Úprava počátečního řešení pomocí mřížky.*

	Velikost obrázku v pixelech			
Název metody	**Typ Odchylky**	800x600	950x684	1280x800
Compute Clustering	Ob	0,0027	0,0027	0,0028
	Pb [%]	4,738	3,640	2,402
Update Vertices	Ob	0,0017	0,0019	0,0019
	Pb [%]	10,421	11,501	11,546

Tab. 4.5.4: *Velikost odchylek především Pb. Z tabulky je vidět jejich výrazné snížení u metody ComputeClustering() a u metody UpdateVertices() její mírné zlepšení.*

A nyní opět proveďme celkový souhrn dosažených výsledků s původními výsledky pro lepší srovnání. Tedy nejprve naměřené časy tří nejdůležitějších metod. *ComputeClustering(), UpdateVertices(), GeneraInitialSolution()* a nakonec změříme celkový čas.

Tabulka 4.5.5 ukazuje výrazné urychlení metody *GeneratelInitialSolution()* pro náhodný výběr. U mřížky je tato metoda okamžitě. U metody *ComputeClustering()* se rychlosti původní a řešení s náhodným výběrem téměř rovnají. U mřížky se tato metoda mírně zhoršila oproti předchozím dvěma řešením. Určitě by se tato obě řešení měla využít.

Na poslední tabulce 4.5.6 jsou vidět rozdíly mezi *Pb*, kde u metod *ComputeClustering()* dvě poslední řešení přinášejí její významné snížení oproti původnímu řešení. Odchylka u metody *UpdateClustering()* zůstává téměř stejná s mírným zlepšením pro velké obrázky.

	Velikost obrázku v pixelech		
	800x600	950x684	1280x800
Název metody	480000	649800	1024000
Původní řešení			
Update Vertices	266,22	373,52	732,13
Compute Clustering	513,17	701,25	1300,36
GenerateInitial Solution	480,73	1192,4	3238,52
Celkový čas výpočtu [s]	**1260,27**	**2267,46**	**5271,37**
Náhodný výběr clusteru			
UpdateVertices	231,2	351,95	638,45
ComputeClustering	502,9	718,13	1250,38
GenerateInitialSolution	181,57	380,79	1035,19
Celkový čas výpočtu [s]	**915,83**	**1451,09**	**2924,48**
Mřížka 20 pixelů			
UpdateVertices	233,53	354,38	639,05
ComputeClustering	730,26	1136,24	2268,78
GenerateInitialSolution	0,38	0,49	0,92
Celkový čas výpočtu [s]	**964,32**	**1491,35**	**2909,07**

Tab. 4.5.5: *Souhrnná tabulka měřených časů pro různé typy počátečních řešení.*

	Velikost obrázku v pixelech		
Název metody	800x600	950x684	1280x800
Původní řešení			
Compute Clustering [%]	8,658	9,968	13,450
Update Vertices [%]	10,244	12,481	14,238
Náhodný výběr clusteru			
Compute Clustering [%]	4,854	4,181	3,399
Update Vertices [%]	10,180	11,329	11,901
Mřížka 20 pixelů			
Compute Clustering [%]	4,738	3,640	2,402
Update Vertices [%]	10,421	11,501	11,546

Tab. 4.5.6: *Souhrnná tabulka měřených Pb pro různé typy počátečních řešení.*

4. 6 Změny metrik

Další experimenty a měření provedeme v oblasti změn metrik. Experimenty provedeme s manhattanskou a druhou mocninu euklidovské metrikou. Použijeme knihovnu se sníženým počtem iterací na $\sqrt{(7n)}+3000$ (metoda *ComputeClustering()*) a $\sqrt{(5n)}+1000$ (*metoda UpdateVertices()*). Počáteční řešení (metoda *GenerateInitialSolution()*) ponecháme beze změn. Budeme měřit opět změny časů jednotlivých metod, hlavně *ComputeClutering(), UpdateVertices(), GenerateInitialSolutin()* a dále pak odchylky bodů, viz kapitola 3.4 urychlování pomocí změn metrik. Obrázky výsledků změn metrik jsou na přiloženém CD.

V našem prvním experimentu se zaměříme na klasickou euklidovskou metriku bez odmocniny. Výpočet tedy probíhá jednodušeji.

Z tabulky 4.6.1 je vidět, že výpočetní doba metod se výrazně urychlila proti původnímu řešení, viz kapitola 4.1 třetí experiment.

Z tabulky 4.6.2 je vidět, že Podíl počtu bodů s odchylkou je malý a přijatelný.

	Velikost obrázku v pixelech		
	800x600	950x684	1280x800
Název metody	480000	649800	1024000
UpdateVertices	214,85	312,4	578,96
ComputeClustering	460,21	655,94	1147,16
GenerateInitialSolution	20,13	39,98	94,3
gain	0,07	0,13	0,15
Reassign	0	0	0
Celkový čas výpočtu [s]	695,32	1008,61	1820,73

Tab. 4.6.1: *Naměřené časy euklidovské metriky bez odmocniny ukazují výrazné zrychlení.*

		Velikost obrázku v pixelech		
Název metody	**Typ odchylky**	800x600	950x684	1280x800
Compute Clustering	Ob	0,0028	0,0017	0,0040
	Pb [%]	2,301	3,402	4,354
Update Vertices	Ob	0,0010	0,0025	0,0018
	Pb [%]	2,880	2,958	2,735

Tab. 4.6.2: *Naměřené odchylky metod. Z tabulky je vidět malá hodnota Pb.*

Poslední experiment provedeme s manhattanskou metrikou. Výpočty jsou zbaveny odmocnin a mocnin oproti euklidovské metrice. Viz kapitola 3.4 urychlování pomocí změn metrik.

Z tabulky 4.6.3 je vidět, že metrika nepřináší žádné výrazné očekávané urychlení. Je spíše ještě pomalejší než samotná euklidovská viz kapitola 4.2 třetí měření.

Z tabulky 4.6.4 vidíme, že odchylky jsou velmi podobné verzi s euklidovskou metrikou viz kapitola 4.2 třetí měření.

	Velikost obrázku v pixelech		
	800x600	950x684	1280x800
Název metody	480000	649800	1024000
UpdateVertices	247,87	358,05	660,6
ComputeClustering	517,01	736,68	1317,83
GenerateInitialSolution	696,64	840,52	4168,52
gain	0,08	0,14	0,15
Reassign	0	0	0
Celkový čas výpočtu [s]	1461,69	1935,55	6147,28

Tab. 4.6.3: *Naměřené časy manhattanské metriky. Z výsledku je vidět, že druhý obrázek je pro manhattanskou metriku vhodnější, má lepší barevné rozložení pixelů. Metoda ComputeClustering() na něm provádí méně přesunů.*

		Velikost obrázku v pixelech		
Název metody	Typ Odchylky	800x600	950x684	1280x800
Compute Clustering	Ob	0,0044	0,0040	0,0060
	Pb [%]	10,149	5,490	17,217
Update Vertices	Ob	0,0024	0,0026	0,0029
	Pb [%]	12,545	9,224	16,077

Tab. 4.6.4: *Naměřené odchylky manhattanské metriky.*

A nyní opět, proveďme celkový souhrn dosažených výsledků i s ještě původními výsledky pro lepší srovnání. Tedy ukážeme si naměřené časy tří nejdůležitějších metod. *ComputeClustering(), UpdateVertices(), GenerateInitialSolution().* Nakonec změříme celkový čas a nezapomeneme přidat jejich odchylky.

Tabulka 4.6.5 Ukazuje souhrnné výsledky prvních tří metod. Z tabulky je vidět, že největší urychlení přináší euklidovská metrika bez odmocniny. Její výsledky vypadají natolik pěkně, že požijeme v kapitole 4.9 její doporučená nastavení.

Tabulka 4.6.5 Ukazuje souhrnné výsledky odchylek metod *ComputeClustering()* a *UpdateVertices()*. Z tabulky je vidět, že poměr počtu bodů s odchylkou je u manhattanské metriky trosku horší než u původního řešení. Ale u euklidovské metriky bez odmocniny je výrazně lepší. *Ob* jsou jsou přibližně stejně velké. Jen manhattanská metrika je má přibližně o 30% vyšší.

Velikost obrázku v pixelech			
	800x600	950x684	1280x800
Název metody	480000	649800	1024000
Původní řešení s euklidovskou metrikou			
Update Vertices	266,22	373,52	732,13
Compute Clustering	513,17	701,25	1300,36
GenerateInitial Solution	480,73	1192,4	3237,52
Celkový čas výpočtu [s]	**1260,12**	**2267,16**	**5270,01**
První řešení s euklidovskou metrikou bez odmocniny			
UpdateVertices	214,85	312,4	578,96
ComputeClustering	460,21	655,94	1147,16
GenerateInitialSolution	20,13	39,98	94,3
Celkový čas výpočtu [s]	**695,32**	**1008,61**	**1820,73**
Druhé řešení s manhattanskou metrikou			
UpdateVertices	247,87	358,05	660,6
ComputeClustering	517,01	736,68	1317,83
GenerateInitialSolution	696,64	840,52	4168,52
Celkový čas výpočtu [s]	**1461,69**	**1935,55**	**6147,28**

Tab. 4.6.5: *Souhrnná tabulka s časy jednotlivých metrik. Jako nejrychlejší metrika se zde ukazuje druhá mocnina euklidovské metriky bez odmocniny*

		Velikost obrázku v pixelech		
Název metody	**Typ Odchylky**	800x600	950x684	1280x800
Původní řešení s euklidovskou metrikou				
Compute Clustering	Ob	0,0029	0,0035	0,0040
	Pb [%]	8,658	9,968	13,450
Update Vertices	Ob	0,0017	0,0019	0,0019
	Pb [%]	10,244	12,481	14,238
První řešení s euklidovskou metrikou bez odmocniny				
Compute Clustering	Ob	0,0028	0,0017	0,0040
	Pp [%]	2,301	3,402	4,354
Update Vertices	Ob	0,0010	0,0025	0,0018
	Pb[%]	2,880	2,958	2,735
Druhé řešení s manhattanskou metrikou				
Compute Clustering	Ob	0,0044	0,0040	0,0060
	Pb [%]	10,149	5,490	17,217
Update Vertices	Ob	0,0024	0,0026	0,0029
	Pb [%]	12,545	9,224	16,077

Tab. 4.6.6: *Souhrnná tabulka ukazující odchylky metod.*

4. 7 Vliv volby parametrů na výsledek

Parametr „Vážení jasu“ ovlivňuje citlivost změny v barvě, tj. při malé citlivosti se projeví jen velká změna. Naopak při velkém důrazu na citlivost v barvě se projeví i malé odchylky v barvě.

Parametr „Velikost shluků“ mění velikost shluků, čím vyšší hodnota, tím jsou shluky „vetší“. Větší shluky umožňují větší kompresi, menší zachovávají lepší kvalitu.

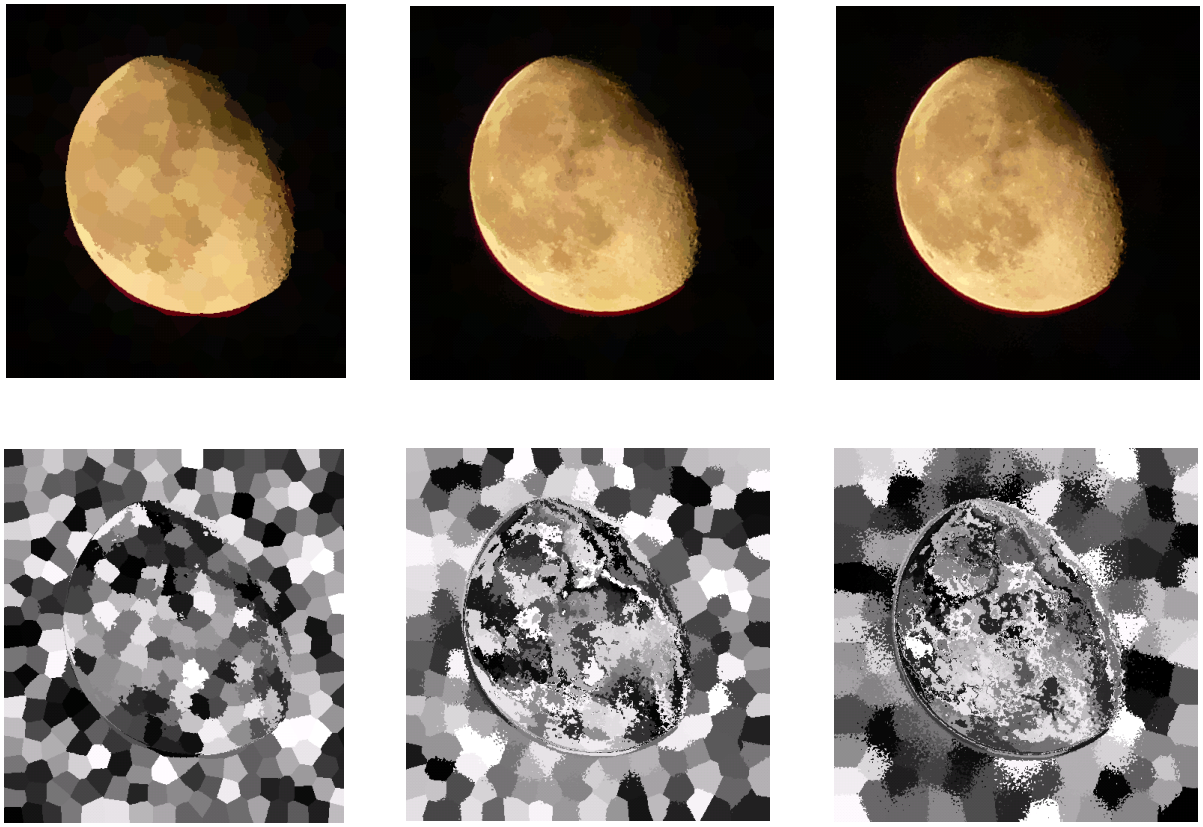
Použijeme původní verzi programu s klasickou euklidovskou metrikou. V následujících ukázkách jsou shluky, které jsou vybarvené konstantní barvou odpovídající středu (nebo průměru) shluku.

Na obrázku 4.7.1 je zachycen Měsíc z původní fotografie bez použití metod shlukování. Tato fotografie je vhodným příkladem pro ukázky vlivu nastavení parametrů na výsledek.

Obr. 4.7.1: *Měsíc bez použití shlukování*

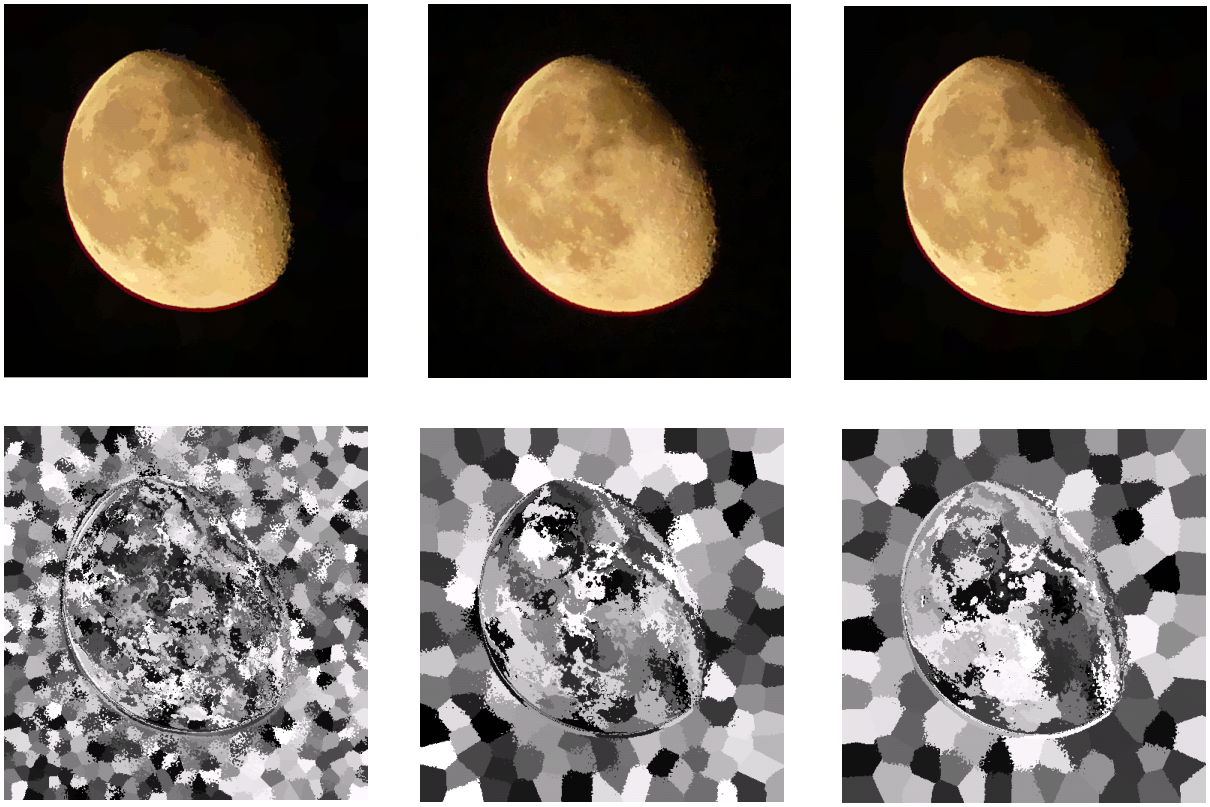
Výsledky chování parametrů jsou ukázány v Obr. 4.7.2(a), 4.7.3(a), 4.7.2(b) … 4.7.3(c).

Obrázky, které k sobě patří jsou vždy pod sebou. První experiment tj. 4.7.2(a)...(c) je s konstantní velikostí shluků 6. Druhý experiment tj 4.7.3(a)...(c) je s konstantní váhou jasu 4



(a) V*áha jasu = 1* **(b)** V*áha jasu = 4* **(c)** V*áha jasu = 8*

Obr. 4.7.2: *Barevný obrázek ukazuje postupné zobrazování detailů při větší váze jasu. Šedotónový zobrazuje, jak se shluky mění podle jasu.*



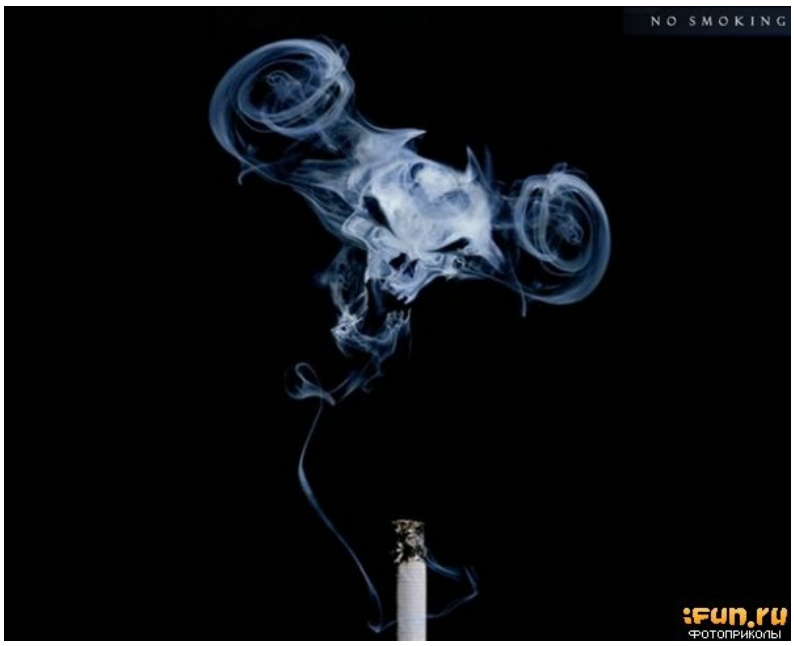
(a) *Velikost shluků = 1* **(b)** *Velikost shluků = 8* **(c)** *Velikost shluků = 12*

Obr. 4.7.3: *Barevný obrázek ukazuje postupné mírné ubývání detailů. Šedotónový zobrazuje, jak se shluky mění a zvětšují podle jasu.*

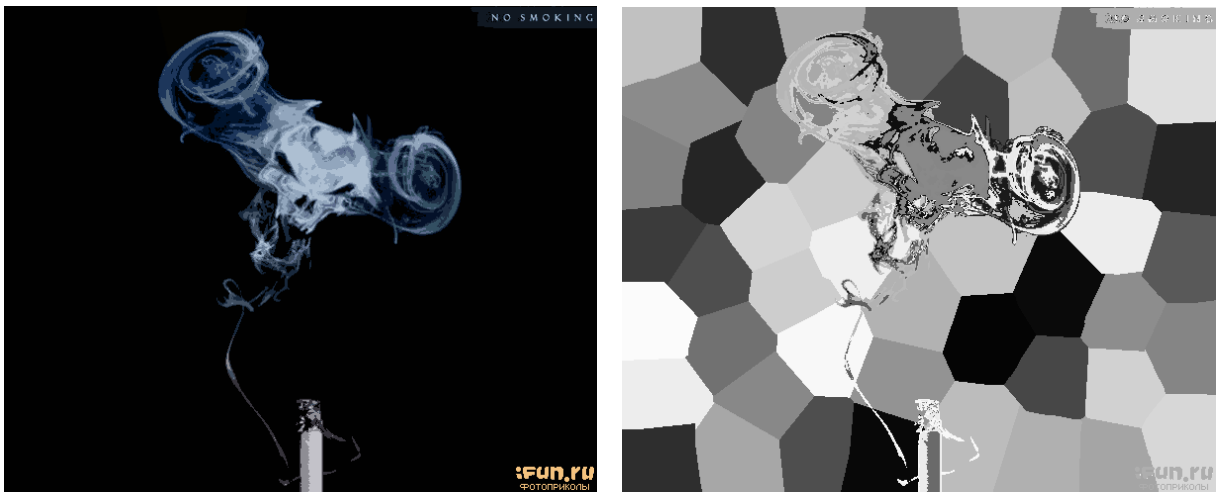
4. 8 Doporučené nastavení parametrů pro Euklidovskou metriku

Velmi záleží, pro jaký účel máme parametry nastavit, tj. jaký obrázek bude zpracováván. Těžko se hledá nějaké ideální řešení pro např. písek s budovami nebo keři, stromy atd. Shlukování větších ploch se stejnou barvou nebývá většinou žádný vetší problém, snad jen nastavit dostatečně velké shluky. Ale ukažme si to na obrázcích.

Obrázek 4.8.1 výstižně pojmenovaný *no smoking* má pro naše účely vhodně velkou plochu. Tento obrázek je původní bez úprav shlukováním. Aplikujeme na něj metody shlukování s parametry Váha jasu: 5 a Velikost shluků: 77. Toto ukazuje obrázek 4.8.2 Na barevném je vidět mírné zhoršení detailů. Ale na šedotónovém vidíme velké shluky, které ohraničuje změna barvy. Ještě pro úplnost, přidejme porovnání s původním obrázkem jako tzv. PSNR: 31.50 [dB], které ukazuje, že se od sebe původní a shlukovaný obrázek příliš neliší.



Obr. 4.8.1: *Ukazuje původní obrázek.*



Obr. 4.8.2 (a) a (b): *Obrázky ukazují shlukování plochy se stejnou barvou.*

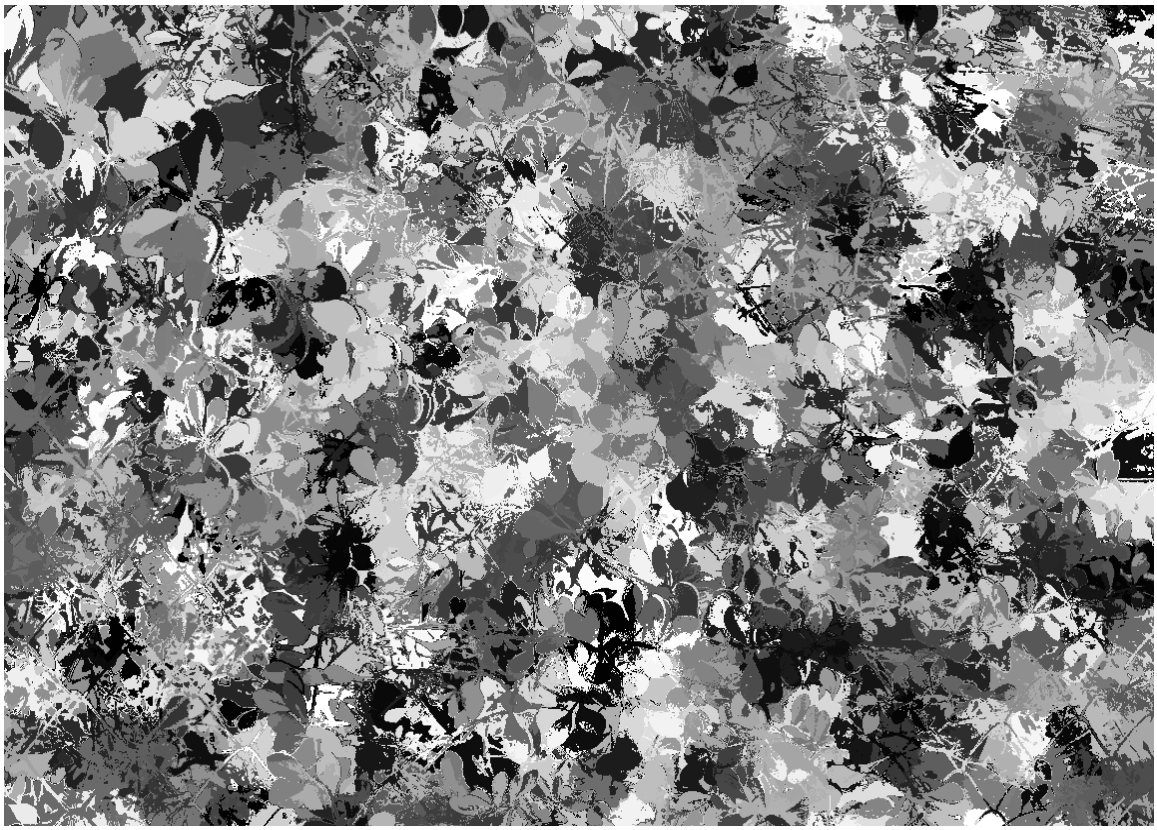
Pro oddělení obrysů a částí větších celků je dobré používat nastavení někde kolem: „Jas 1-4“ a velikost shluků volit co možná největší, aby se spojily pixely, které mají podobnou nebo stejnou barvu. Je tedy vhodné nastavit „Velikost shluků 7-15“ dle velikosti ploch.

Takové nastavení je vhodné pro segmentaci nebo velkou kompresi.

A nyní si takové nastavení ukažme na dvou obrázcích. Tedy nejprve původní obrázek bez užití shlukování. Na obrázku 4.8.3 je výřez z fotografie keře v plotu, ale pro náš účel bude stačit.

Obr. 4.8.3: *Původní fotografie keře v plotě.*

Dále tedy zkusme nastavit parametry shlukování takto: Váha jasu: 1 a Velikost shluků: 7. Opět s fotografií keře. Toto nastavení parametrů zachycuje obrázek 4.8.4. Na takto zvolených parametrech je dobře vidět, jak nám nastavení „vytahuje“ listy keře. Doplňme si opět pro tento obrázek PSNR: 20.39 [dB]. Není to příliš velká shoda, ale pro účely segmentace a oddělení objektů je dostačující.



Obr. 4.8.4 *Barevný obrázek ukazuje „vytažení“ lístků se spojením do jedné barvy nebo dvou barev. Šedotónový zobrazuje obrysy shluků.*

Jako druhý obrázek použijeme výřez z fotografie borovice, tedy kousek větve s jejími jehlicemi. Tento původní je zachycen na obrázku 4.8.5.

Obr. 4.8.5: *Původní fotografie jehlic borovice.*

Zkusme nastavit parametry trochu jinak než v předchozím případě. Nastavme shlukování takto: Váha jasu: 2 a Velikost shluků: 11. A opět s původní fotografií jehlic borovice . Toto nastavení parametrů zachycuje obrázek 4.8.6. Na takto zvolených parametrech je dobře vidět, jak nám nastavení „vytahuje“ jehlice a pozadí splývá. Opět zhodnoťme, jak moc se obrázek liší od původního: PSNR 25.82 [dB], tento výsledek je pěkný na to, že jsme „pouze“ chtěli segmentaci obrazu.

Obr. 4.8.6 (a): *Obrázek v barevné verzi ukazuje „vytažení“ jehlic.*

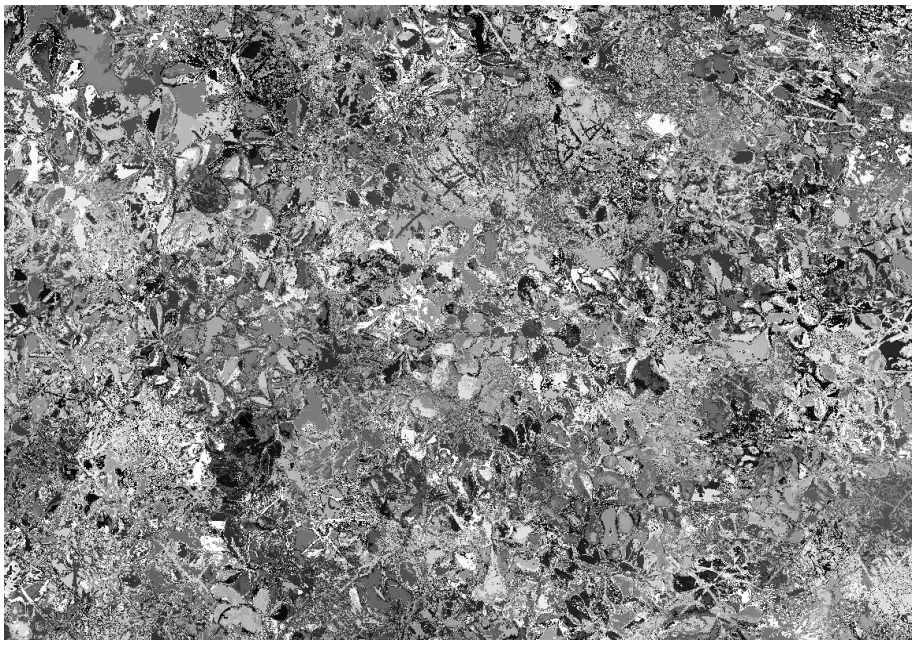
Obr. 4.8.6 (b): *Obrázek v šedotónové verzi ukazuje, jak se shluky rozloží podle jehlic.*

Dále se budeme věnovat zvýšení detailů obrázku. Toho dosáhneme zvýšením parametru Váha jasu přibližně kolem (4-10) a velikosti shluků přibližně (6-12). Ukažme si to opět na obrázcích.

Znovu použijeme původní dvě fotografie, také proto, abychom mohli lépe porovnat rozdíly mezi těmito nastaveními.

Nastavme parametry shlukování takto: „váha jasu“: 6 a „velikost shluků“:6.Výsledek je vidět na obrázku 4.8.7, kde PSNR: 23.72 [dB] by sice mohlo být větší,ale okem téměř nelze poznat rozdíl od původní fotografie, která je na obr. 4.8.3. U verze (b) je vidět rozdrobení shluků.

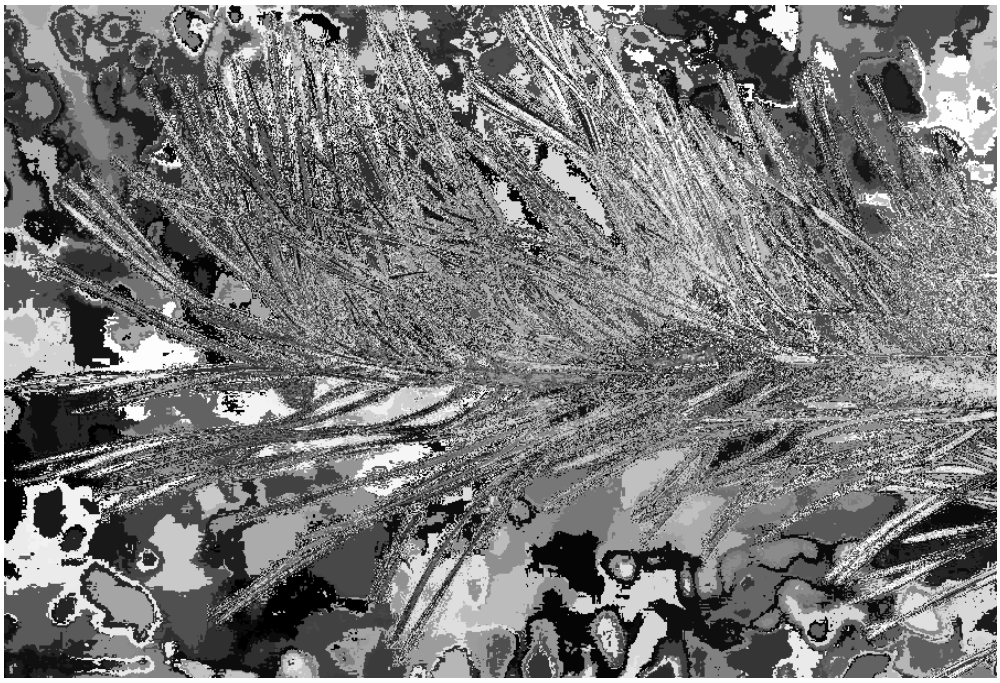
Obr. 4.8.7 (a): *Obrázek ukazuje, že již nelze téměř poznat rozdíl od původní fotografie.*



Obr. 4.8.7 (b): *Obrázek ukazuje rozdrobení shluků.*

Zkusme opět změnit parametry shlukování takto: „váha jasu“:12 a „velikost shluků“: 13, jak ukazuje obrázek 4.8.8, kde PSNR: 29.72 [dB] se jeví dostatečné a okem se jen těžko hledá viditelný rozdíl oproti původní fotografii, která je na obrázku 4.8.5. U verze (b) je vidět rozdrobení shluků, ale stále ještě drží pěkně tvar podle jehlic.

Obr. 4.8.8 (a): *Ukazuje detaily jehlic.*



Obr. 4.8.8 (b): *Obrázek ukazuje rozdrobení shluků, ale stále ještě výrazně oddělené jehlice.*

A nyní proveďme, jak je dobrým zvykem, závěrečnou sumarizaci zjištěných poznatků. Pro přehlednost v několika tabulkách.

Pro obrázky s rozsáhlými plochami bychom měli volit přibližně takováto nastavení, která ukazuje tabulka 4.8.1.

Velikost shluků	60-100	70-120
Váha jasu	4-7	6-8

Tab. 4.8.1: *Tabulka s nastavením shlukování pro velké plochy.*

Pro obrázky s oddělením objektů např. listů, jehlic, menších ploch s podobnou barvou atd. Je dobré volit nastavení parametru podle tabulky 4.8.2.

Velikost shluků	11-15	14-17
Váha jasu	1-3	3-4

Tab. 4.8.2: *Tabulka s nastavením shlukování pro oddělení objektů.*

Pro lepší kvalitu a zvýrazněné detaily v obrázcích, by se „Váha jasu“ a „Velikost shluků“ neměly příliš od sebe lišit, ale je dobré volit váhu jasu o něco menší, jak ukazuje tabulka 4.8.3.

Velikost shluků	6	7	8	9	10	11	12
Váha jasu	4-6	5-7	6-8	7-9	8-10	9-11	10-12

Tab. 4.8.3: *Tabulka s nastavením shlukování pro výrazné detaily.*

Pokud jsou výsledky málo kvalitní, což by se při těchto nastavení nemělo stávat (viz předchozí tabulka 4.8.3), můžeme volit jas o malinko vyšší, ale je třeba opatrně, aby se výsledky (shluky) příliš nerozdrobily. Tím bychom neušetřili žádnou informaci jako se tomu děje např. u drobných zrnek písku, soli, atd. Jak je uvedeno v následující tabulce 4.8.4.

Velikost shluků	6	7	8	9	10	11	12
Váha jasu	7-8	8-9	9-10	10-11	11-12	12-13	13-14

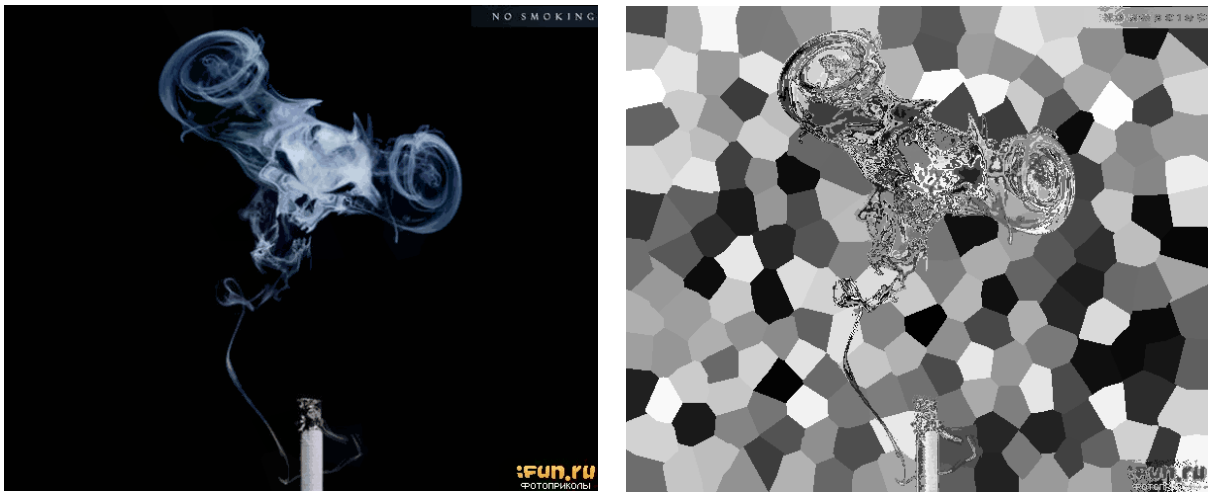
Tab. 4.8.4: *Tabulka s nastavením shlukování pro zvýraznění detailů.*

Další výsledky testovacích obrázků jsou na přiloženém CD.

4. 9 Doporučené nastavení parametrů pro euklidovskou metriku bez odmocniny.

Experimenty provedeme se stejnou množinou obrázku jako v kapitole 4.8.

Na obrázku 4.9.1 (viz původní obr. 4.8.1) je shlukování velkých ploch. Parametry shlukování jsme zvolili takto: Váha jasu: 5 a Velikost shluků 500. Uveďme si PSNR, které je 36.20 [dB]. Plyne z něj velká shoda obrázků. Na obrázcích je vidět, že velikost shluků by mohla být ještě i o něco větší.



Obr. 4.9.1 (a) a (b): *Obrázky ukazují shlukování velkých ploch. Na šedotónovém (b) je vidět, jak jsou poskládané v ploše .*

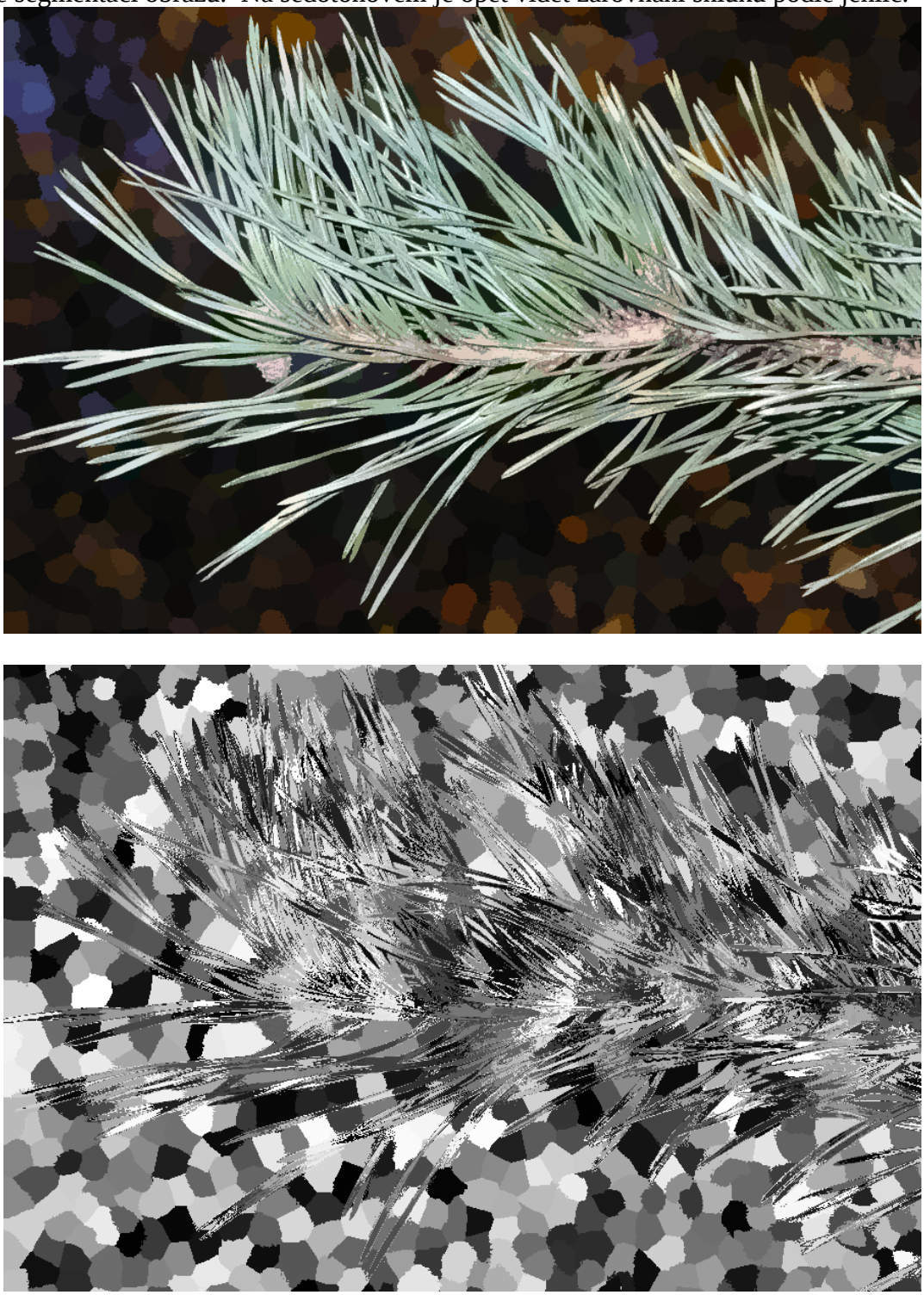
Pro oddělení obrysů a částí větších celků je dobré používat nastavení někde kolem: „Jas 1-2“ a velikost shluků volit co možná největší, aby se spojily pixely, které mají podobnou nebo stejnou barvu. Je tedy vhodné nastavit „Velikost 90-200“ dle velikosti ploch.

Takové nastavení je vhodné pro segmentaci nebo velkou kompresi.

Zkusme tedy nastavit parametry shlukování takto: Váha jasu: 1 a Velikost shluků 91. Opět s fotografií keře, viz obrázek 4.8.3. Toto nastavení parametrů zachycuje obrázek 4.9.2. Na takto zvolených parametrech je dobře vidět, jak nám nastavení „vytahuje“ listy keře. Uveďme opět PSNR, jehož hodnota je 22.44 [dB]. Hodnota PSNR není příliš vysoká (plyne zní odlišnost obrazů), ale potřeby segmentace je dostačující. Na šedotónovém Obr. je vidět, jak se shluky natočí podle lístků.

Obr. 4.9.2 (a) a (b): *Barevný obrázek ukazuje „vytažení“ lístků se spojením do jedné barvy nebo dvou barev. Šedotónový zobrazuje obrysy shluků a jejich zarovnání podle lístků.*

Pro fotografii z obrázku 4.8.5 zkusme nastavit parametry úplně stejně jako v předchozím případě. Nastavme shlukování takto: Váha jasu: 1 a Velikost shluků: 91. Toto nastavení parametrů zachycuje obrázek 4.9.3. Na takto zvolených parametrech je dobře vidět, jak nám nastavení „vytahuje“ jehlice a pozadí splývá. Opět srovnejme odlišnost od původní fotografie výpočtem PSNR: 26.40 [dB], plyne z něj dobrá shoda obrázků, byt jsme chtěli pouze segmentaci obrazu. Na šedotónovém je opět vidět zarovnání shluků podle jehlic.



Obr. 4.9.3 (a) a (b): *Horní barevný obrázek (a) ukazuje vytažení jehlic a splývání pozadí. Šedotóvý ukazuje protažení shluků do směru jehlic.*

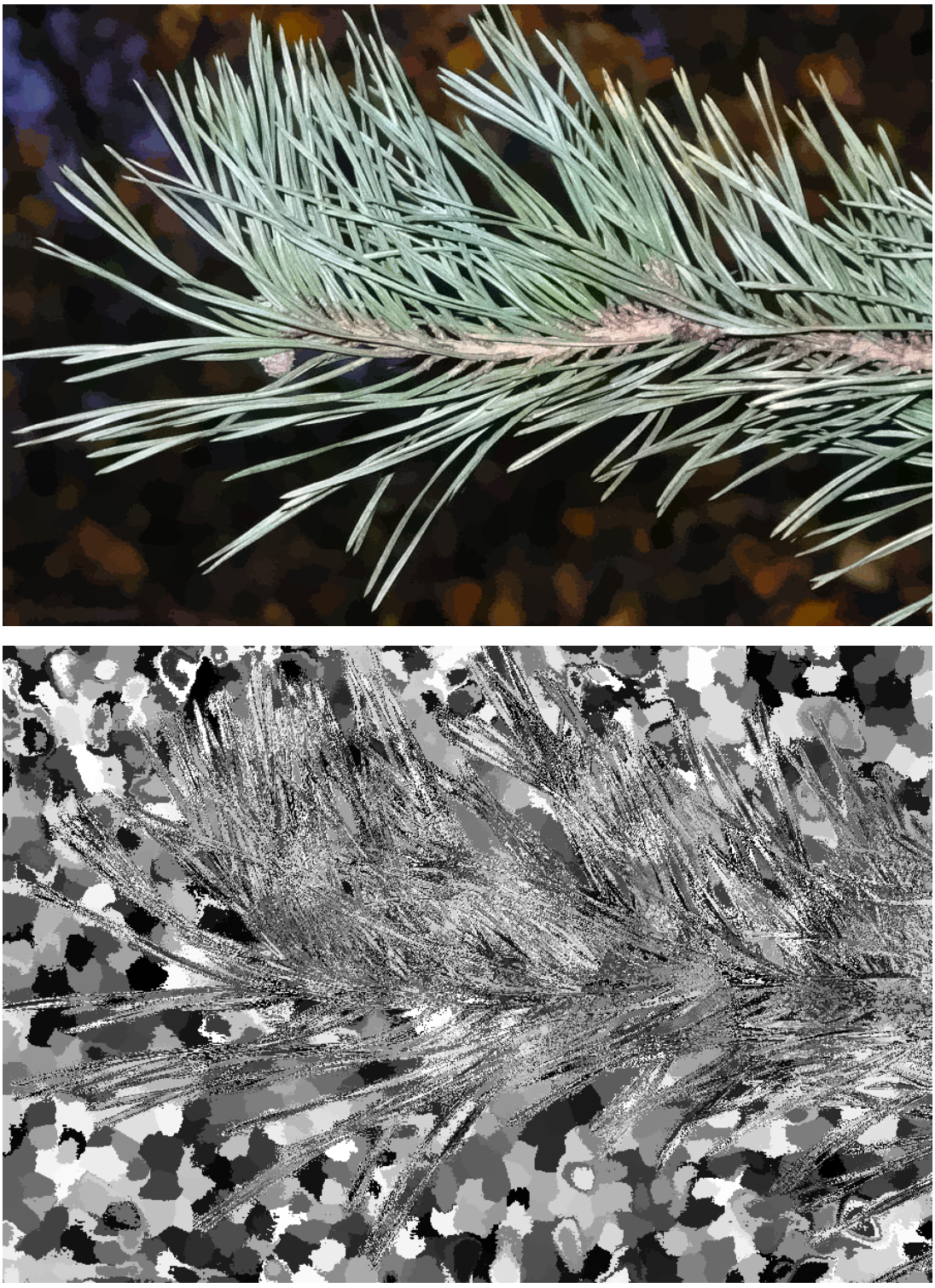
Dále se budeme věnovat zachycení detailů obrázku. Toho dosáhneme zvýšením parametru Váha jasu přibližně v intervalu (3-5) a velikosti shluků přibližně v rozsahu (60-90). Jak to vypadá si ukažme na několika obrázcích.

Znovu použijeme původní dvě fotografie, viz obrázky 4.8.3 a 4.8.5, také proto, abychom mohli lépe porovnat rozdíly mezi těmito nastaveními pro obrázky.

Zkusme tedy nastavit parametry shlukování takto: Váha jasu: 3 a Velikost shluků 82. Podívejme se co nám o obrázků 4.9.4 říká PSNR: 24.99 [dB]. Jeho hodnota by mohla být ještě o trochu vyšší, ale opět okem nelze nelze téměř poznat viditelný rozdíl oproti původní fotografii 4.8.3 U verze (b) dochází k rozdrobení shluků.

Obr. 4.9.4 (a) a (b): *Obrázek (a) ukazuje, že již téměř nemůžeme poznat žádný rozdíl od původní fotografie. Šedotónový ukazuje rozdrobení shluků.*

Zkusme použít stejné nastavení parametrů, jako v předchozím případě. Na obrázku 4.9.5 (a) již téměř prakticky není poznat rozdíl od původní fotografie, která je na Obr. 4.8.5, jak nám potvrzuje PSNR: 30.24 [dB]. U verze (b) dochází k rozdrobení shluků podle mírných odchylek v barvě. Avšak stále je vidět soudržnost shluků podél jehlic.



Obr. 4.9.5 (a) a (b): *Na obrázku (a), barevná verze vidíme krásné detaily jehlic. Obrázek (b) šedotónová verze ukazuje rozdrobovaní shluků podle barev.*

A nyní shrňme zjištěné poznatky v několika tabulkách.

Pro obrázky s rozsáhlými souvislými plochami bychom měli volit přibližně nějaká takováto nastavení, která ukazuje tabulka 4.9.1. Tato metrika není příliš vhodná por použití na velkých plochách, vzhledem k tak velkému parametru „Velikost shluků“

Velikost shluků	500-800	700-1000
Váha jasu	3-5	5-6

Tab. 4.9.1: *Tabulka s nastavením shlukování pro velké plochy.*

Pro obrázky s oddělením objektů, např. listů, jehlic, menších ploch s podobnou barvou atd., je dobré volit nastavení parametru podle tabulky 4.9.2.

Velikost shluků	90-150	140-200
Váha jasu	1-2	3

Tab. 4.9.2: *Tabulka s nastavením shlukování pro oddělení objektů.*

Pro lepší kvalitu a zvýraznění detailů v obrázcích, by se „Váha jasu“ a „Velikost shluků“ neměly nastavovat podle tabulky 4.9.3.

Velikost shluků	60-80	70-100	90-140	130-200
Váha jasu	3-4	4-5	5	6

Tab. 4.9.3: *Tabulka s nastavením shlukování pro výrazné detaily.*

Pokud jsou výsledky málo kvalitní, což by se při těchto nastavení nemělo stávat (viz předchozí tabulka 4.9.3), můžeme volit jas o malinko vyšší, ale je třeba opatrně, aby se výsledné shluky příliš nerozdrobily, viz tabulka 4.9.4.

Velikost shluků	60-80	70-100	90-140	130-200
Váha jasu	4-5	5-6	6-7	7-8

Tab. 4.9.4: *Tabulka s nastavením shlukování pro zvýraznění detailů.*

Další výsledky experimentů jsou na přiloženém CD.

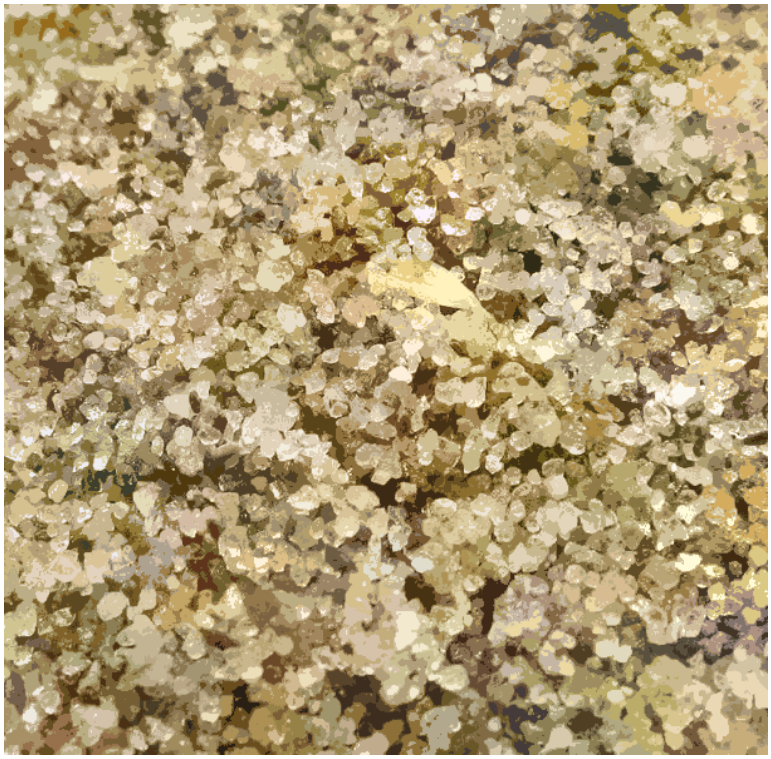
4.10 Selhávání metody

Jedná se především o drobná zrnka písku, soli atd., kde je téměř nemožné nastavit tak malé shluky a při tom získat nějakou úsporu. Na obrázcích je mnoho drobných detailů, žádné větší souvislé plochy, ze kterých by mohly vzniknout shluky. Takové obrázky je obtížné komprimovat, protože „skoro každý pixel je důležitý“.

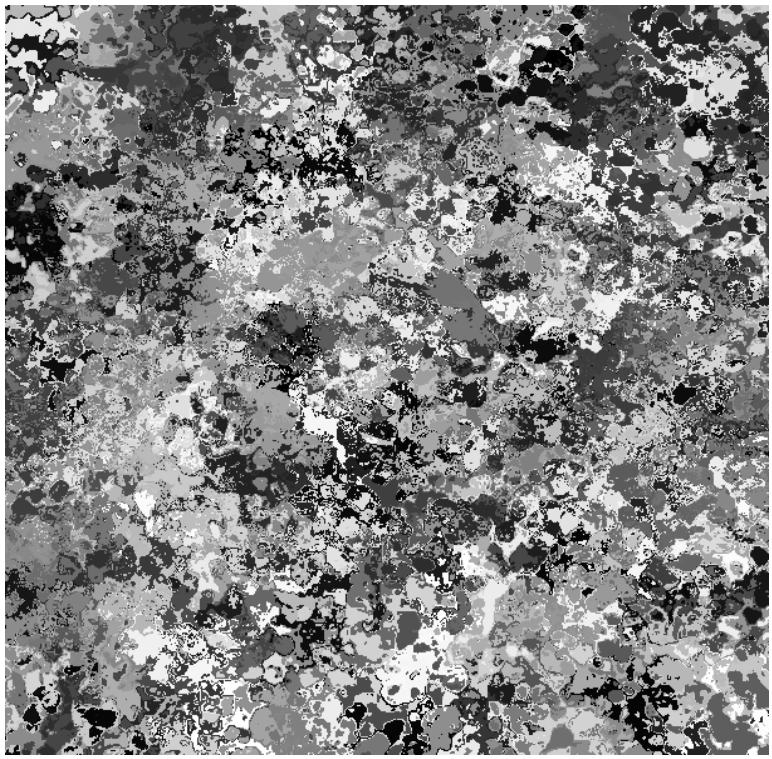
Ukažme si to opět na příkladu. K tomuto účelu použijeme obrázek písku, který je na obrázku 4.10.1. Tento obrázek ukazuje spoustu drobných zrnek písku, u kterých jen stěží nastavíme nějaké dobré parametry shlukování.

Obr. 4.10.1: *Na obrázku jsou vidět drobná zrnka písku.*

Pokusme se aplikovat shlukování na takovýto obrázek. Zvolme parametry shlukování: Váha jasu: 2 a Velikost shluků 12. Výsledek zachycuje obrázek 4.10.2. Barevný obrázek (a) vykazuje velké ztráty barvy, což by ale nevadilo, pokud chceme zjistit tvar zrn. Ale v šedotónovém Obr. (b) nedokážeme rozpoznat jednotlivá zrna, z tohoto pohledu naše metoda selhává.



Obr. 4.10.2 (a): *Barevná verze obrázku po aplikaci shlukování.*



Obr. 4.10.2 (b): Šedotónová *verze obrázku po aplikaci shlukování. Je zde vidět spíše změť koleček, zohýbaných ploch, nic, co by připomínalo oddělená zrna.*

4.11 Spojování podobných shluků

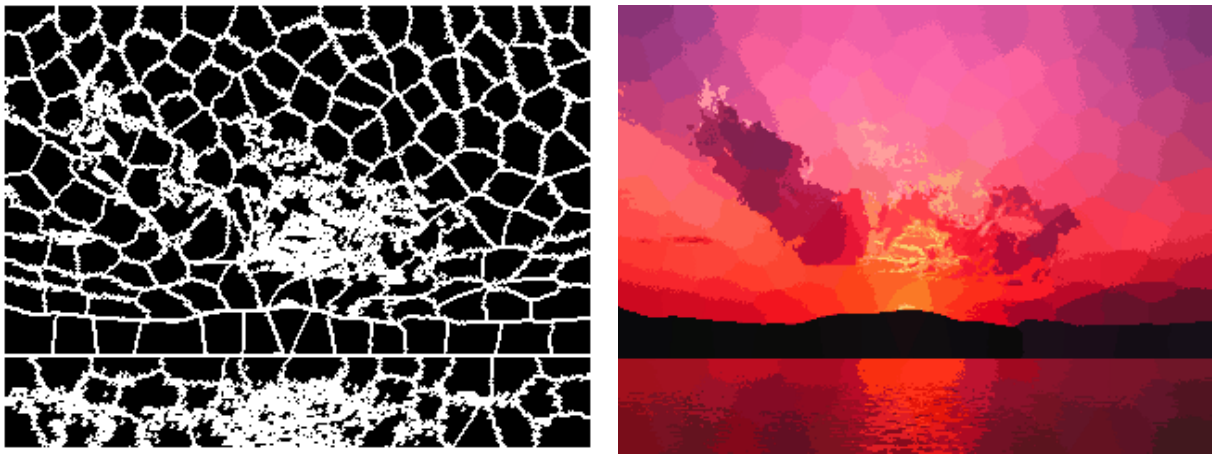
Spojování shluků s podobnou barvou umožňuje lepší oddělení objektů při segmentacích, dále zvýšení kompresních poměrů pro obrázek.

Maska obrázku je výstupní soubor, ve kterém jsou pouze dvě barvy (černá a bíla). Maska tedy obsahuje bíle hranice shluků, které jsou bílou barvou, ostatní plochy jsou černě. Ukažme si opět, jak toto spojování shluků probíhá a co vlastně dostaneme za výsledky.

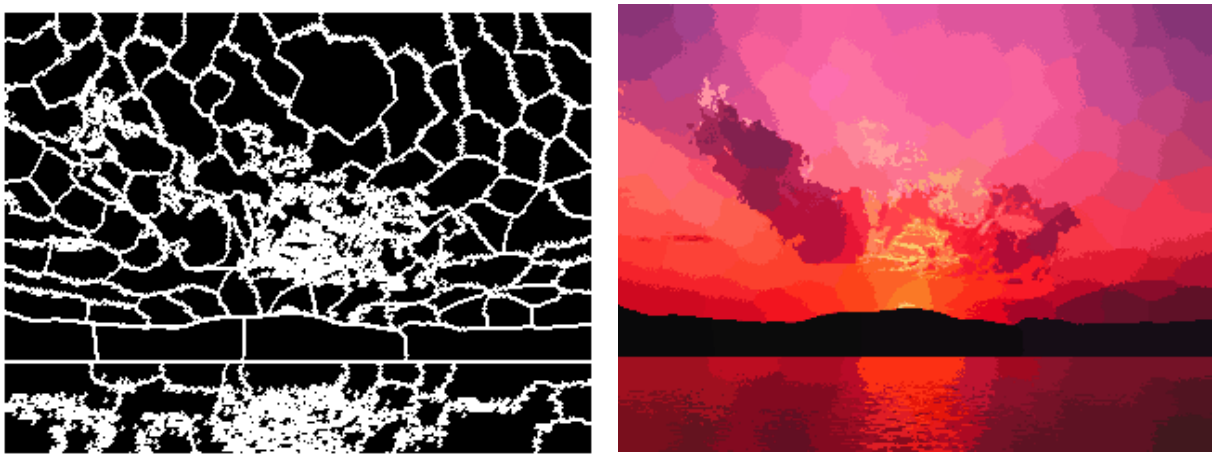
Obrázek 4.11.1 (a) a (b) ukazuje použití masky bez použití citlivosti na barvu. Tedy shluky zatím nebudeme spojovat. (a) je černobílá kompresní verze a (b) je testovací, ukazují se na ní průměry barev.

Nastavme citlivost na změnu v barvě na hodnotu např. 9. Tento postup je vidět na obrázku 4.11.2 (a) a (b). Je tu ukázáno počínající spojování shluků. V obrázku masky (a) je vidět, jak se shluky spojují do větších celků a v obrázku (b), jak se barvy průměrují podle ploch shluků.

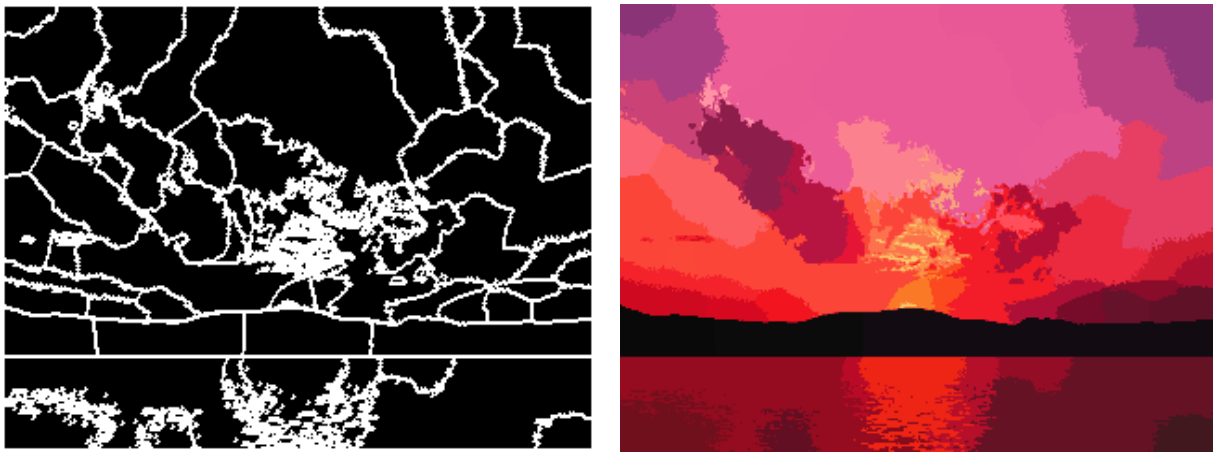
Zkusme ještě nastavit citlivost na změnu v barvě na vysokou hodnotu 30, což je vidět na obrázku 4.11.3 (a) a (b). (a) ukazuje výrazné spojování shluků, které mají velké plochy. Obrázek (b) ukazuje průměrování takovýchto ploch stejnou barvou.



Obr. 4.11.1 (a) (b): *Obrázek masky západu slunce bez užití spojování shluků. (a) je černobílá verze, (b) barevná.*



Obr. 4.11.2 (a) (b): *Obrázek masky západu slunce s použitím spojování shluků. (a) je černobílá verze, vidíme, jak se shluky zvětšují. V obrázku (b) v barevné verzi se spojené shluky zprůměrovaly a vykreslily touto barvou.*



Obr. 4.11.3 (a) (b): *Obrázek masky západu slunce s velkým použitím spojování shluky. (a) je černobílá verze, vidíme velké shluky . U (b), tedy barevné verze se vytvořily velké oblasti s průměrnou barvou.*

Masku lze využít pro rekonstrukci obrázku s použitím metody RBF od Ing. Jiřího Zapletala. Ale vzhledem k rozsáhlosti práce nebyla implementována. Výsledky jejího použití na obrázcích lze najít na přiloženém CD spolu s dalšími výsledky spojování shluků.

4.12 Shrnutí experimentů

Pokusme se na závěr celé kapitoly shrnout naše pozorování, experimenty a měření.

Původní počet iterací tj. *0,1*n* je již prakticky nepoužitelný pro obrázky větší než 0,5 mil. pixelů. Časová náročnost je více než 1,7h. Z kapitoly 4.1 a 4.2 plyne, že lze aplikaci efektivně urychlit. I pro velké obrázky může běžet v ještě přijatelném čase. Efektivně využitelné se zdají být počty iterací *0,04*n* nebo $\sqrt{(7n)}+3000$. U počtu iterací, který je dán předpisem *0,04*n*, lze zaručit *Pd* přibližně do 7%. Ale i tak je možné, že délka výpočtu bude dlouhá (3,7h), jedná se o obrázky s velikostí přes 1 mil. bodů. U počtu iterací zvolených takto $\sqrt{(7n)}+3000$ je kladen důraz na vysokou rychlost, ale s možností větší odchylky. Nízkou odchylku již nelze zaručit. Pozvolna s velikostí obrázku roste. Ale jsme schopni shlukovat i obrázky větší než 1 mil. pixelů, v ještě přijatelném čase přibližně kolem 1,5h a více pro ještě větší obrázky. Logaritmický počet iterací tj. $počet\ \ iterací = 100 * \log_2(n) + 2000$ již skutečně podle naměřených odchylek není vhodný. Nicméně takto zvolený počet iterací dokáže shlukovat v podstatě jakékoliv velikosti v přijatelném čase (1,3h). Ale vzhledem k odchylkám jsme ho zavrhli.

Z kapitoly 4.3 plyne, že pro konstantní odchylku potřebujeme mít počet iterací lineárně závislý na počtu bodů obrázku. To ale znamená, že při zachování konstantní odchylky jen stěží budeme moci shlukovat velké obrázky v rozumném čase.

Z kapitoly 4.4 je vidět, že s rostoucím počtem iterací *Pd* klesá, ale ne nějak výrazně. Na grafu je vidět, že se vyplatí počet iterací definovat přibližně takto: *<0,03-0,06>*n*. Vyšší počet iterací nepřináší již tak významné zlepšení a spíš vede ke zpomalení výpočtu.

Z měření času vyplynula potřeba ještě nějak urychlit počáteční řešení. Jak je vidět z kapitoly 4.5, vyplatilo se to. Počáteční řešení je v případě vybrání 20% shluků náhodně přibližně 3x rychlejší než původní řešení i pro velké obrázky, bez zhoršení odchylek. Na metodu *ComputeClustering()* takováto změna neměla žádný negativní vliv. Výsledkem je běh rychlejší až o 40% a pro ještě větší obrázky bude ještě vyšší. Další experiment se týkal použití mřížky. Z měření plyne, že počáteční řešení je okamžitě pro jakýkoliv velký obrázek. Ale jaký vliv to bude mít na metodu *ComputeClustering()*? Ukázalo se, že tato metoda dělá pro velké obrázky 2x tolik práce než dříve. Ale opět celkový čas nutný k běhu výpočtu vychází

dobře. Téměř totožně, jako v případě vybrání 20% shluků náhodně, tj. 50 minut pro obrázky s velikostí 1 mil. pixelů

Po provedení těchto úprav knihovny můžeme konstatovat, že jsme schopni shlukovat obrázky větší než 1 mil. bodů v ještě použitelném čase. Obrázky bývají velké, nejsou jen „pokusné a malé“. Pro aplikaci komprese je potřeba počítat s reálnými a velkými fotografiemi, které již značně přesahují 1 mil. pixelů.

Další urychlení jsme prováděli v oblasti použití jiných metrik. Velkým zklamáním je pro nás manhattanská metrika, která by měla být rychlejší než euklidovská, avšak výsledky měření této metriky nepřinášely žádné očekávané zrychlení, dokonce vedly k mírnému zpomalení. Na druhou stranu druhá mocnina euklidovské metriky přinesla výrazné urychlení. Oproti původní euklidovské metrice s odmocninou je asi 2 až 2,5x rychlejší. Tato metrika by si jistě zasloužila využití v praxi.

Další částí, která si zaslouží pozornost, je chování shlukování při změně parametrů. Při zvětšování parametru *Velikost shluků* se shluky rozšiřují a zvětšují. Jejich natáčení, protažení podle barev vytvářejí segmentaci objektů, jejich velikost lze využít pro úsporu informace, kterou je nutné uložit, tedy vytváří kompresi. Naopak při větším jasu, tedy důrazu na barvu, se projeví i malý rozdíl v barvě. Taková nastavení nejsou příliš vhodná k segmentacím či kompresím a vedou k drobení shluků. Avšak zachovávají věrně obrázek.

Z kapitol 4.8 a 4.9 se podařilo získat doporučená nastavení pro shlukování digit. obrázků. Jejich podrobnosti nalezneme v tabulkách 4.8.1 až 4.9.4.

Porovnání obou variant euklidovské metriky je následující. Z experimentů vyplývá lepší použitelnost klasické euklidovské metriky na obrázky se velkými plochami, kde se mohou tvořit velké shluky. K segmentacím objektů se hodí obě metriky stejně. A nakonec k detailům v obrázku vycházejí obě metriky též přibližně stejně. Možná trochu lépe (vyšší PSNR) druhá mocnina euklidovská metriky.

Metoda je nevhodná pro shlukování malých zrn písku, soli a podobných obrázků, kde každý pixel je důležitý, se jen těžko hledá nějaké dobré nastavení. Shluky se musí tvořit malé, ale tím nedojde k úspoře oproti původnímu obrázku. Na takovýchto obrázcích metoda selhává, ale vcelku očekávaně.

Nakonec shrňme možnosti komprese při užití spojování podobných shluků a tvorbě masky. Tímto způsobem můžeme spojit i shluky, které se liší třeba jen nepatrně v barvě, ale jsou natolik rozlehlé, že je samotné shlukování nedokázalo s námi nastavenými parametry spojit. Spojování podobných shluků můžeme opět využít pro oddělení objektů.

5. Zhodnocení

Původně sloužila knihovna pro zpracování bodů ve 3D. Nové použití vyžaduje shlukování pixelů, které mají souřadnice x, y a barvu R, G, B. Dále pak najít přepočet barvy a vzdálenosti dvou bodů. Poté vyzkoušet na mnoha obrázcích, jak se shlukovací knihovna chová a doporučená nastavení parametrů, se kterými ji můžeme nastavovat. Celkově ji urychlit změnou výpočtu počátečního řešení, použitím mřížky a s použitím vhodně zvolených bodů obrázku. Dále pak urychlit s použitím menšího počtu iterací, vyzkoušet její chování na jiných metrických prostorech. Použití manhattanské a euklidovské metriky bez odmocniny. A na závěr umožnit spojování shluků s podobnou barvou.

Knihovnu se podařilo převést na funkčnost pro digitalizovaný obraz za pomoci výpočtu rozdílu v barvě RGB a následně za pomoci kombinace rozdílu v barvě a vzdálenosti bodů.

Na mnoha experimentech jsme dokázali najít doporučené nastavení parametrů, tj. Váhy jasu a Velikosti shluků. Nastavení pro klasickou euklidovskou metriku a druhou mocninu euklidovské metriky jsou jednoduchá a nejlépe je ukáže tabulka 5.1, která zjednodušeně shrnuje výsledky experimentů.

		klasická euklidovská	Druhá mocnina euklidovská
Plocha	Velikost shluků	60-120	500-1000
	Váha jasu	4-8	3-6
Segmentace	Velikost shluků	11-17	90-200
	Váha jasu	1-4	1-3
Detaily	Velikost shluků	6-12	60-200
	Váha jasu	4-12	3-6

Tab. 5.1: *Zjednodušené nastavení parametrů pro knihovnu.*

Dále se aplikaci podařilo výrazně urychlit, a to nejenom v oblasti snižování počtu iterací, ale také změnami v počátečním řešení.

Urychlení v oblasti snížení počtu iterací: Knihovna v původní verzi jen stěží dokázala v rozumném čase shlukovat velký obrázek (1 mil. bodů), trvalo jí to přibližně 7,5h, což vůbec nebyl dobrý výsledek. Povedlo se nám díky snížení počtu iterací jít přibližně k 1,4h. To již je poměrně velký úspěch, za cenu větších Pb. Ale i nadále je to dlouhá doba. Je možné ještě další urychlení, tentokrát v oblasti generování počátečního řešení. Můžeme zvolit mřížku nebo výběr náhodně zvolených shluků, obojí dává přibližně stejný čas a to přibližně 50 minut pro obrázek (1 mil pixelů). To je velmi příjemný výsledek.

Také použití jiné metriky přineslo své výsledky (druhé mocniny Euklidovské). Výpočet s použitím této metriky běžel pouhých 31 minut oproti původní verzi s Euklidovskou metrikou.

Podařilo se spojování shluků s podobnou nebo stejnou barvou, tím pádem můžeme spojovat rozsáhlé plochy a ušetřit tak informaci při kompresích.

Nepovedlo se najít dobrá nastavení parametrů pro shlukování písku, zrn atd. Protože veliké množství velmi malých shluků nepřináší žádnou výhodu. Dále mahattanská metrika nepřinesla žádné očekávané urychlení programu, proto jsme ji zavrhli.

Použitá literatura

[1] http://cs.wikipedia.org/wiki/Prahování
[2] http://en.wikipedia.org/wiki/Color_quantization
[3] http://en.wikipedia.org/wiki/Segmentation_(image_processing)
[4] http://cs.wikipedia.org/wiki/Barevný_histogram
[5] Tomáš Mikolov. Color Reduction Using K-Means Clustering, s. 1-3
[6] J. Skála, I. Kolingerová. Clustering Geometric Data Streams. In SIGRAD 2007, s. 17-23, 2007.
[7] http://cs.wikipedia.org/wiki/Metrika

Příloha č. 1: Shlukování - Uživatelská dokumentace

Pro běh programu potřebujeme .NET Framework 2.0.
Po spuštění programu nejsou přístupná všechna tlačítka.
Dostupná tlačítka po spuštění jsou pouze v menu:

Open picture - Otevře vstupní obrázek (*.bmp, *.png, *.jpg atd.), který je pak zobrazen vlevo.

Open text file - Otevře a zobrazí (vykreslí) vstupní textový soubor vpravo, viz obrázek 1.1. Červeně je obarven střed shluku. Body patřící do shluku mají stejnou barvu. Tato barva je volena náhodně. Formát souboru, který je schopen program načíst ,vypadá takto:
22,20; // šířka , výška obrázku
2,14; 2,11;0,7;3,14; // souřadnice x,y středu shluku; mezera body, které patří do shluku x,y;
1,2; 1,1;2,2;1,3;20,17;
Samotný střed shluku může a nemusí být mezi body uveden.

Po otevření obrázku, tj. Open picture, se zpřístupní další tlačítko.

Compute clustering - Spočte shlukování a zobrazí výsledek vpravo. Barvu výsledku je možné ovlivňovat přepínači *Cluster Centre* (střed shluku) nebo *Cluster Average* (průměr shluku), tj. buď je shluk obarven podle středu shluku nebo jeho průměru. Dále lze tento proces ovlivňovat parametry *Weight of brightness* (Váha jasu) a *Size of clusters* (velikost shluků), změnami v editačních polích.

Weight of brightness - ovlivňuje citlivost změny v barvě, tj. při malé citlivosti se projeví jen velká změna. Naopak při velkém důrazu na citlivost v barvě se projeví i malé odchylky v barvě.

Size of clusters - mění velikost shluků, čím vyšší hodnota, tím jsou shluky „vetší“. Větší shluky umožňují větší kompresi, menší zachovávají lepší kvalitu.

Pro nastavení těchto parametrů je dobré se podívat do kapitoly 4.8. Doporučené nastavení pro euklidovskou metriku.

A nyní již jsou k dispozici všechny možnosti programu, jak v menu tak i na formuláři, viz Obr. 1.2.

Save picture - Uloží dva výstupní obrázky (výsledek shlukování) do souborů, a to jak barevnou tak i šedotónovou kombinaci shlukování. V kombinaci je možné lépe detekovat shluky, barva je volena náhodně.

Popis výstupních obrázků:
(př. *Západ slunce_3_14.png Západ slunce_256_3_14.png)*
První soubor bez „256“ je barevný, první parametr 3 udává nastavení váhy jasu ,viz výše. Další parametr „14“ udává nastavení velikosti shluků.
Druhý soubor s „256“ je šedotonový, následují stejné parametry, jako u prvního souboru.

Mask - Umožňuje spojit některé shluků. Má jeden parametr *Sensitivity on change of colour* (citlivost na změnu barvy). Při vyšší hodnotě můžeme spojit i méně podobné shluky, které se více liší v barvě. Výstupem jsou dva obrázky barevný a černobílý. Barevný ukazuje spojování shluků. Na černobílém obrázku, kde bílé plochy jsou hranice shluků a naopak černé plochy jsou vnitřky shluků.

Popis výstupních obrázků:
(př. *Západ slunce_maska_9.png Západ slunce_test_9.png)*
První soubor s označením „maska“ je černobílý, parametr: 9 udává citlivost na změnu barvy.
Druhý soubor s označením „test“ je barevný, parametr: 9 udává citlivost na změnu barvy.

Cluster to file - uloží výsledky shlukování do textového souboru. Formát výstupního textového souboru:

22,20; // šířka , výška obrázku, jakožto první parametry souboru
2,14; 2,11;0,7;3,14; //souřadnice x,y středu shluku; mezera body, které patří do shluku x,y;
1,2; 1,1;2,2;1,3;20,17;
Střed shluku (přestože také patří do shluku) se mezi body už neopakuje.

A na závěr si ještě popišme jednotlivé přepínače:
Cluster centre – nastaví barvu celého shluku podle jeho středu.
Cluster average – nastaví barvu celého shluku podle průměru pixelů, které obsahuje.

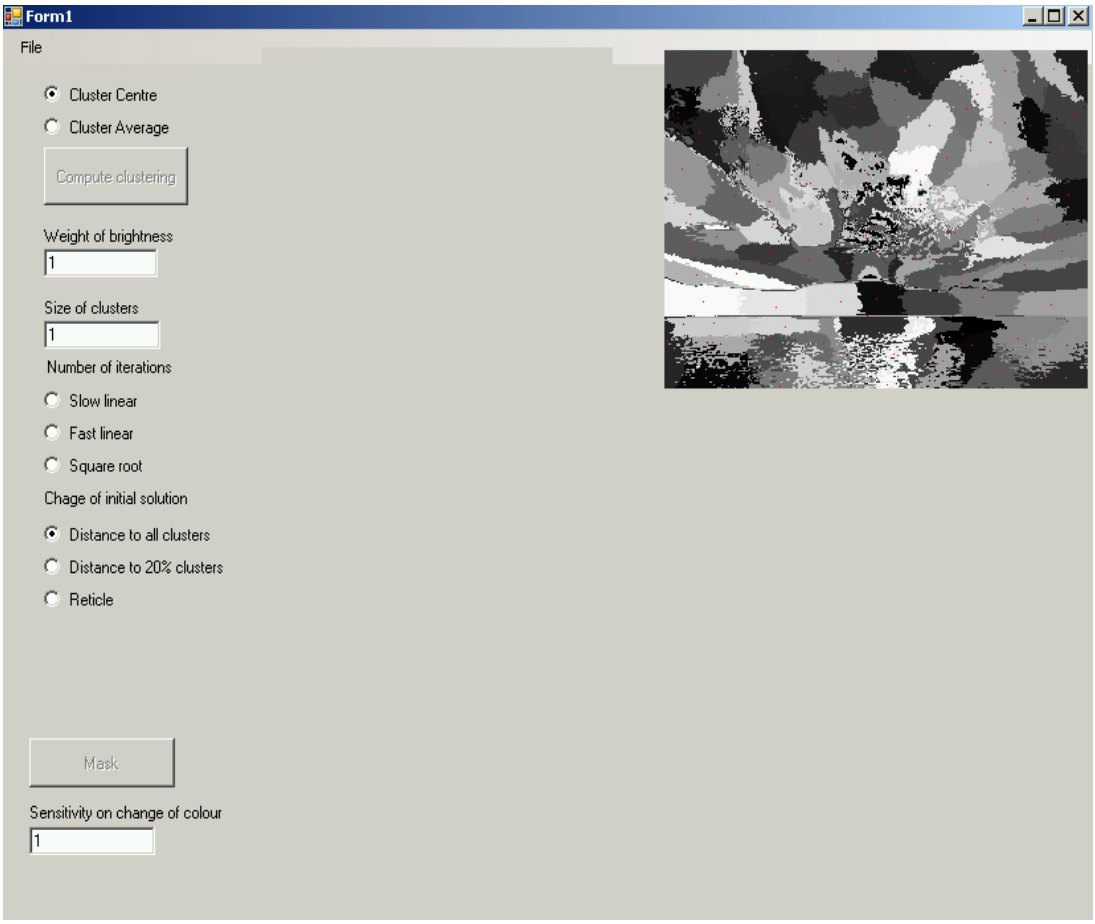
Přepínače, které mění rychlost běhu programu:

Number of iterations – mění počet iterací (opakování)

- ***Slow linear*** - Nastaví počet opakování na „velmi důkladně“. Ale tato nastavení mohou mít neblahý vliv na délku běhu programu. Proto je třeba toto políčko volit uváženě a pro velké obrázky se mu vyhnout.
- ***Fast linear*** - Nastaví počet opakování na „středně důkladně“. Volba je vhodná pro středně velké obrázky. Ale i této se může stát, že výpočet poběží delší čas. Pro extrémně velké obrázky.
- ***Square root*** - Nastaví počet opakovaní na „rychlý běh“. Tato volba umožňuje shlukování extrémně velkých obrázků v ještě přijatelném čase.

Change of intial solution – mění způsob výpočtu počátečního řešení.

- ***Distance to all clusters*** - „Pomalé“ generování počátečního řešení.
- ***Distance to 20% clusters*** - „Středně“ rychlé generování počátečního řešení.
- ***Grid*** - „Rychlé“ generování počátečního řešení.



Obr. 1.1: *Ukázka, jak vypadá zobrazený soubor, ve kterém jsou uložené shluky.*

Obr. 1.2: *Nápověda k ovládání programu. Vlevo je originální obrázek. Vpravo je výsledek shlukování.*

Příloha č. 2: Vstupní testovací obrázky a fotografie

Obr. Příl. 2: *Testovací vstupní obrázky: Západ slunce, ZČU v létě.*

Obr. Příl. 2: *Testovací vstupní obrázky:Tráva (pýr).*