SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STAVEBNÁ FAKULTA

AUTOMATICKÉ RETUŠOVANIE POŠKODENÝCH OBRÁZKOV

BAKALÁRSKA PRÁCA

Študijný program: Matematicko-počítačové modelovanie

Študijný odbor: Aplikovaná matematika

Vedúci záverečnej práce:

Mgr. Mariana Remešíková, PhD.

Autor bakalárskej práce:

Ján Prosuch

Bratislava 2010

Čestne prehlasujem, že som bakalársku prácu Automatické retušovanie poškodených obrázkov vypracoval samostatne s použitím citovanej literatúry a s odbornou pomocou vedúceho práce.

.....

Ďakujem vedúcej svojej práce, Mgr. Mariane Remešíkovej PhD., za nesmiernu ochotu, neoceniteľné rady a hlavne za čas a trpezlivosť.

Abstrakt

Cieľom tejto práce bolo vytvoriť softvér na úpravu poškodených obrázkov a fotografií. Konkrétne sme sa zamerali na automatické retušovanie implementáciou jednej z moderných metód zaoberajúcich sa touto problematikou - Fast Marching metódy.

Pri implementácií sme testovali jednotlivé zložky ovplyvňujúce metódu za účelom čo najlepších výsledkov. Takisto sme sa snažili odhaliť nedostatky metódy a následne nájsť najvhodnejšie využitie.

Abstract

The goal of our work was to develop a software tool for the inpainting of damaged pictures and photos. Specifically, we focused on the automatic retouching by implementation of one of the newest methods – the Fast Marching method.

In the scope of our work we tested various factors affecting the method. We also tried to identify the drawbacks and then find the best possibilities for the aplication of the method.

Obsah

Úvod 1

1. Program Fotošop junior 2

2. Použité metódy a funkcie 4

2.1 Bresenhamov algoritmus 4

2.2 Scanline algoritmus 6

2.3 Distance funkcia 7

2.4 Fast Marching metóda 7

2.5 Inpainting metóda 9

3. Implementácia 15

4. Pozorovanie a testovanie 16

5. Záver 24

Zoznam použitej literatúry 25

Úvod

V čase rozkvetu moderných technológií, kedy už aj tí najmenší sú vybavení mobilnými telefónmi s možnosťou fotografovania a vďaka dokonalejším fotoaparátom, ktoré dodajú pocit profesionality každému, sa stala práca s fotografiami veľkým hitom. Ide však zväčša len o amatérske pokusy s mnohými nedostatkami, ktoré sa dnes ale dajú odstrániť pomocou rozličných softvérov.

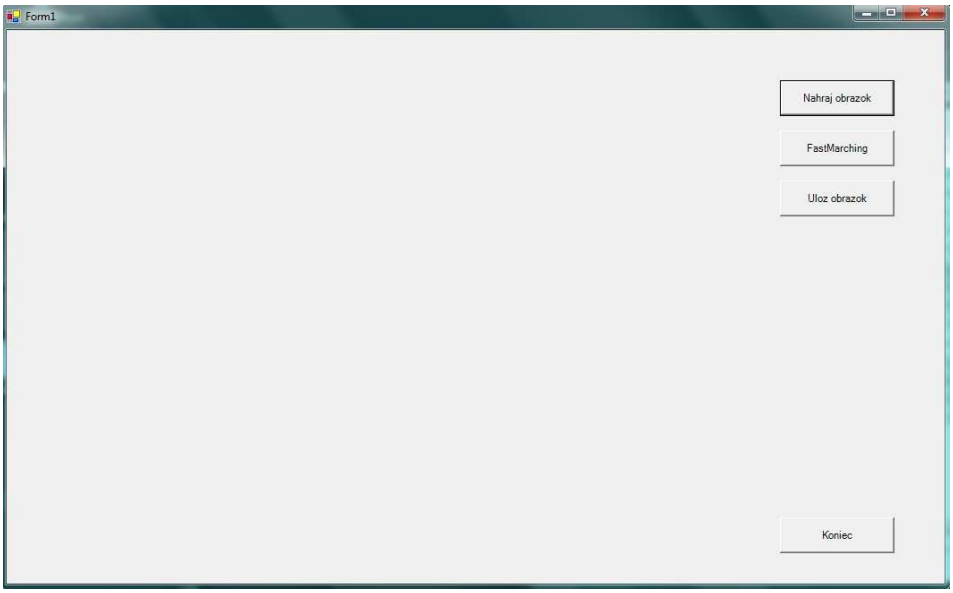
My sme sa zamerali na jeden z týchto problémov, konkrétne ide o automatické retušovanie poškodených obrázkov. Jedná sa o úpravu obrázkov a fotografií, ktoré sú znehodnotené rôznymi škrabancami a škvrnami. Retušovanie sa však môže rozšíriť aj na obrázky, ktoré vo svojej podstate nie sú poškodené, ale pre rôzne publikačné alebo umelecké účely potrebujú byť upravené. Najjednoduchšia metóda riešiaca túto problematiku funguje spôsobom, že sa najprv vyberie poškodená oblasť a následne sa určí farba, ktorou sa oblasť prekryje. Takáto metóda je však nedokonalá a efektívne využitie je len na menších, homogénnych oblastiach.

Na tému inpainting bolo publikovaných množstvo článkov, napr. [2], [4], [6], ktoré využívajú rôzne dômyselnejšie metódy, často založené na numerickom riešení diferenciálnych rovníc.

My sme vytvorili softvér implementujúci modernú metódu retušovania s využitím Fast Marching metódy, ktorá slúži na riešenie eikonalovej rovnice. Metóda bola publikovaná v článku [1]. Následne sme sledovali jej možnosti a zisťovali jej využiteľnosť. Najdôležitejšími faktormi boli najmä kvalita opravených oblastí a rýchlosť realizácie. Softvér ma názov Fotošop junior a bol vytvorený v prostredí Visual C++.

1. Program Fotošop junior

Program Fotošop junior bol vytvorený v jazyku C++ a implementovaný v prostredí Visual C++ 2003. Jeho ovládanie je veľmi jednoduché a intuitívne, keďže obsahuje iba základné funkcie, ako sú načítanie obrázku, jeho úprava a následne aj možnosť upravený obrázok uložiť. Zamerali sme sa hlavne na implementáciu danej metódy, preto vytvorený softvér v aktuálnej verzii nie je ošetrený pre obrázky väčšie ako 800x600 pixlov.

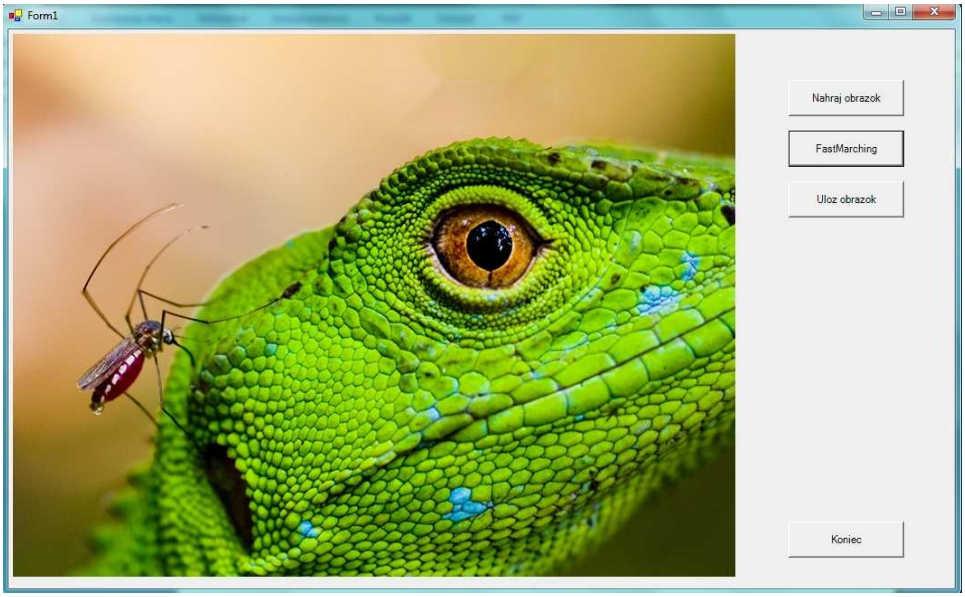


Obr. 1: *Program po spustení*

Obr. 2: *Program po načítaní obrázka* [13]

Prvým krokom pri práci s programom je načítanie obrázku. Program rozoznáva iba dva formáty a to *.jpg a *.bmp. Bez načítania sa dá vyznačiť oblasť, na ktorej má program pracovať, ale samotná metóda neprebehne, pretože potrebuje hodnoty o pixloch z obrázka. Po načítaní je nutné vybrať oblasť, ktorá sa bude retušovať, klikaním myšou po pracovnej ploche a následne sa pomocou tlačidla FastMarching spustí proces retušovania. Čas, za ktorý metóda prebehne závisí od veľkosti vyznačenej plochy. Pri menších oblastiach je program schopný opraviť chybu do niekoľkých sekúnd. Pri väčších oblastiach to trvá niekoľko desiatok sekúnd. Ak je výsledok vyhovujúci, môže sa uložiť.

Obr. 3: *Program s vyznačenou oblasťou* [13]



Obr. 4: *Program po odstránení loga* [13]

2. Použité metódy a funkcie

Na úpravu obrázkov bol použitý algoritmus retušovania podľa [1] využívajúci metódu Fast Marching [5]. Tento algoritmus pracuje vo vyznačenej oblasti a pre svoje správne fungovanie potrebuje poznať vnútorné body tejto oblasti a takisto body jej hranice. Hraničné body sa získavajú rasterizáciou úsečiek ohraničujúcich oblasť pomocou Bresenhamovho algoritmu. Vnútorné body sa získavajú pomocou vypĺňacieho algoritmu scanline fill. V nasledujúcich častiach popíšeme všetky potrebné metódy.

2.1 Bresenhamov algoritmus

Bresenhamov algoritmus slúži na rasterizáciu úsečky, teda nájde množinu pixlov, ktoré najlepšie zodpovedajú danej úsečke. Bol vyvinutý Jackom E. Bresenhamom v roku 1962. Algoritmus určuje, ktorý bod v rastri sa má vykresliť tým, že sa vyberie pixel, ktorý leží najbližšie k analyticky vyjadrenej úsečke. Algoritmus využíva iba celočíselnú matematiku, preto je veľmi efektívny. Popis algoritmu nájdeme v [7], [9].

Majme analyticky vyjadrenú úsečku rovnicou:

$$y = \frac{\Delta y}{\Delta x} x + q,$$

kde x $\in$ < x_0, y_0> a y $\in$ < x_1, y_1 >. Ďalej uvažujme dva body A_1:(x_0, y_0) a A_2:(x_1, y_1), pre ktoré platí: A_2 je napravo od A_1 a pre smernicu:

$$m = \frac{\Delta y}{\Delta x},$$

platí 0<m<1. Nech x_0, y_0, x_1, y_1 sú celé čísla a ak nie sú, zaokrúhlime ich. Vysvetlíme si algoritmus pre tento konkrétny prípad. Pre úsečky s inými smernicami platí rovnaký princíp a algoritmus sa líši len v ľahko odvoditeľných detailoch.

Predstavme si, že sme už vybrali a vykreslili pixel so súradnicami (x_i, y_i) a chceme zistiť, ktorý pixel bude ďalší v poradí. Vďaka hodnote smernice 0<m<1 bude v X-ovom smere prírastok vždy 1, avšak v Y-ovom smere bude menší ako 1 a teda musíme vybrať z dvoch možností (x_i+1,y_i) alebo (x_i+1,y_i+1). Vyberáme na základe vzdialeností d_1 a d_2

stredov pixlov (x_i+1,y_i) a (x_i+1,y_i+1) od skutočného bodu úsečky (x_i+1, y). Stačí nám vedieť, či ich rozdiel:

$$\Delta d = d_1 - d_2 = 2m(x_1 + 1) - 2y_1 + 2q - 1,$$

je kladný alebo záporný. Ak je kladný, vyberieme bod (x_i+1,y_i+1) a v opačnom prípade vyberieme bod (x_i+1,y_i). Avšak stále tu nevyužívame iba celočíselnú aritmetiku, preto výraz ešte zjednodušíme prenásobením celej rovnice Δx:

$$p_i = \Delta d \Delta x = 2\Delta y(x_i + 1) - 2\Delta x y_i + \Delta x(2q - 1)$$

Predpokladali sme, že A_1 je naľavo od A_2, preto Δx>0 a teda znamienko p_i je rovnaké ako znamienko Δd. Následne vzťah pre p_{i+1} bude vyzerať takto:

$$p_{i+1} = 2\Delta y(x_{i+1} + 1) - 2\Delta x y_{i+1} + \Delta x(2q - 1)$$

Po odčítaní vzťahov pre p_i a p_{i+1} dostávame takéto vyjadrenie p_{i+1}:

$$p_{i+1} = p_i + 2\Delta y - 2\Delta x(y_{i+1} - y_i)$$

Podľa znamienka p_i teda vyberáme aj p_{i+1} nasledujúcim spôsobom. Ak $p_i \leq 0$ vyberáme bod (x_i+1,y_i), y_{i+1}= y_i a teda:

$$p_{i+1} = p_i + 2\Delta y$$

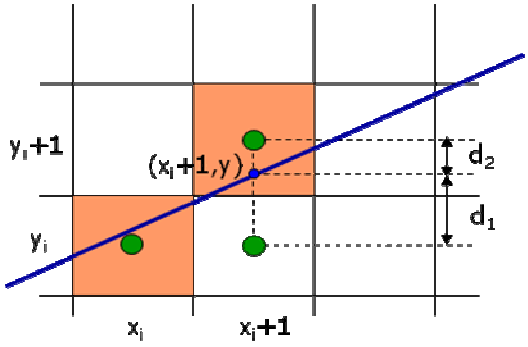
V opačnom prípade vyberieme bod (x_i+1,y_i+1), y_{i+1}= y_i+1 a teda:

$$p_{i+1} = p_i + 2\Delta y - 2\Delta x$$

Hodnotu p_1 dostaneme dosadením x_1 a y_1=mx_1+q do vzťahu pre p_i:

$$p_1 = 2\Delta y - \Delta x$$

a teda počítame iba s celočíselnými konštantami. V ostatných oktantoch funguje algoritmus obdobne, akurát sa mení súradnica, ktorá má konštantný prírastok či úbytok.



***Obr. 5**: Bresenhamov algoritmus*

2.2 Scanline algoritmus

Algoritmus slúži na vypĺňanie polygonálnych oblastí. Pracuje po riadkoch a využíva analytické vyjadrenia úsečiek tvoriacich hranicu oblasti. Popis algoritmu nájdeme v [7], [9].

Pri implementácii je potrebné si vytvoriť tabuľku všetkých hrán, pričom treba skrátiť každú hranu o jeden pixel zdola, aby sme sa vyhli problémom s nejednoznačnosťou, a vynechať vodorovné hrany. Každá hrana je daná X-ovou a Y-ovou súradnicou svojho horného bodu (x_i, y_i), Y-ovou súradnicou dolného bodu y_i^L a obrátenou hodnotou smernice *w*. Následne treba hrany usporiadať primárne vzostupne podľa Y-ovej súradnice y_i, vzostupne podľa X-ovej súradnice x_i a nakoniec zostupne podľa *w*. Ďalej postupujeme po riadkoch od minimálnej Y-ovej súradnice všetkých vrcholov mnohouholníka až po maximálnu Y-ovú súradnicu. V každom riadku sa nájdu priesečníky daného riadku s hranami oblasti a spoja sa po dvojiciach idúcich za sebou úsečkou farby, ktorou sa oblasť vyplní. Aby algoritmus nemusel v každom riadku hľadať priesečníky so všetkými hranami oblasti, je dobré si vytvoriť ešte tabuľku aktívnych hrán, kde budú uložené len hrany, o ktorých sa vie, že majú určite prienik s aktuálnym riadkom. Algoritmus potom zisťuje len priesečníky aktuálneho riadku s hranami z tabuľky aktívnych hrán. Do tejto tabuľky sa hrana dostane vtedy ak začína v aktuálnom riadku, a je z nej vyradená vtedy, ak v aktuálnom riadku končí. Hľadanie priesečníkov sa dá vykonávať

veľmi efektívne. Predstavme si, že chceme nájsť priesečník x_a nejakej úsečky s riadkom $y=y_a$. Keďže už poznáme priesečník x_a-1 s riadkom $y=y_a$-1, môžeme písať:

$$w = \frac{x_a - x_i}{y_a - y_i},$$

$$x_a = wy_a - wy_i + x_i,$$

$$x_{a-1} = w(y_a - 1) - wy_i + x_i = wy_a - wy_i + x_i - w = x_a - w.$$

2.3 Distance funkcia

Distance funkcia k zadanej množine $\Omega_0 \subset R^n$ je funkcia T: $R^n \rightarrow R$, ktorej hodnota sa v každom bode rovná jeho vzdialenosti, v našom prípade euklidovskej, od množiny Ω_0. Pri retušovaní vyznačenej časti obrázka sa postupuje tak, že sa najskôr vypočíta distance funkcia T k hranici vyznačenej oblasti vo všetkých jej vnútorných bodoch. Potom sa jednotlivé body zafarbujú smerom od hranice dovnútra, pričom ako prvý sa vždy zafarbí ten nezafarbený bod, ktorý ma najmenšiu hodnotu T.

Distance funkcia sa dá počítať analyticky, boli však vyvinuté veľmi efektívne metódy, ktoré počítajú distance funkciu numericky, pomocou riešenia eikonalovej rovnice[1]:

$$|\nabla T| = 1, \qquad (1)$$

s podmienkou T=0 na Ω_0. Niektoré spôsoby riešenia tejto rovnice sa dajú nájsť napríklad v [3], [5], [8].

2.4 Fast Marching metóda

Fast Marching metóda bola predstavená Jamesom A. Sethianom [5] ako veľmi efektívna numerická metóda na riešenie eikonalovej rovnice s okrajovými podmienkami.

Ako sme už spomínali, riešenie T rovnice (1) je zobrazenie, ktoré každému pixlu z vyznačenej oblasti Ω priradí jeho vzdialenosť k hranici $\partial\Omega$. To nám vytvorí vrstevnice $\partial\Omega$ idúce smerom do stredu Ω. Normála N k $\partial\Omega$ je rovná presne ∇T. To nám teda zaručuje, že vždy vyfarbujeme vrstvu pixlov, ktorá je najbližšie k už správne zafarbenému okoliu. Hodnota T nám zároveň presne určuje bod, ktorý sa má vyfarbiť, keďže je to vždy ten, ktorý má najmenšiu hodnotu medzi ešte nevyfarbenými pixlami.

Sila a efektívnosť Fast Marching metódy spočíva v tom, že distance funkcia sa nepočíta v každom kroku na celej oblasti, ale výpočty prebiehajú len pre istú podmnožinu bodov. Tieto body tvoria len úzky pás (tzv. narrow band), čo podstatne redukuje množstvo potrebných výpočtových operácií.

Vysvetlime si teraz podrobne princíp Fast Marching metódy. Pre každý pixel si uchováme dve hodnoty – hodnotu distance funkcie T a hodnotu f, ktorá je indikátorom toho, kde sa bod nachádza vzhľadom na výpočtovú oblasť. Body, kde sa distance funkcia zatiaľ nepočítala a bude sa počítať, budú mať označenie f = 1. Rozhranie týchto dvoch oblastí tvoria body, kde sa distance funkcia práve počíta, tie označíme f = 0. Body, ktoré sa nachádzajú v nepoškodenej alebo už správne zafarbenej časti označíme f = 2.

Použitá numerická metóda je založená na nasledujúcej aproximácii eikonalovej rovnice [3], [5].

$$max(D^{-x}T, -D^{+x}T, 0)^2 + max(D^{-y}T, -D^{+y}T, 0)^2 = 1 \qquad (2)$$

kde $D^{-x}T(i, j) = T(i, j) - T(i-1, j)$ a $D^{+x}T(i, j) = T(i+1, j) - T(i, j)$ a obdobne pre y. Táto schéma je vlastne kvadratickou rovnicou s neznámou T(i, j) (distance funkcia T v pixli so súradnicami i, j) za predpokladu, že poznáme niektorú z hodnôt T(i-1, j), T(i+1, j), T(i, j-1), T(i, j+1). Keď sa pozrieme na definície použitých diferencií, vidíme, že vďaka maximám v rovnici (2) môžu na hodnotu T(i, j) vplývať len tie susedné body (i-1, j), (i+1, j), (i, j-1), (i, j+1), v ktorých je hodnota T menšia ako aktuálna hodnota T(i, j). Pritom zmenu môžu prinášať len tie body, v ktorých sa ešte počíta, teda kde ešte nie je f = 2. Preto ak sa dospeje do stavu, že hodnota T(i, j) je menšia ako hodnoty T vo všetkých okolitých bodoch, kde $f \neq 2$, výpočet sa pre pixel (i, j) končí a hodnota T(i, j) je už konečná aproximácia distance funkcie pre tento bod.

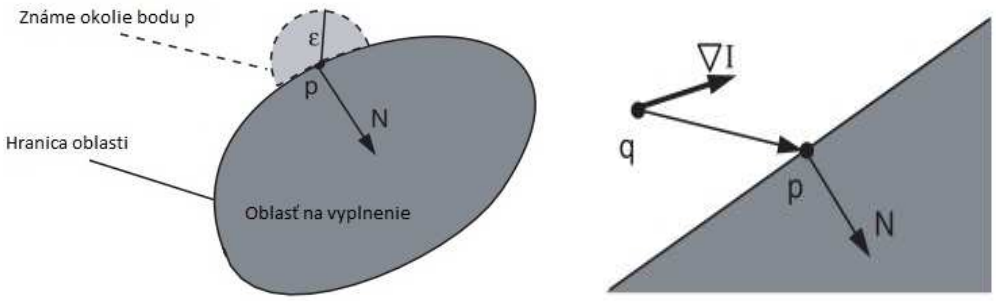
Z tohto dôvodu si body v našom výpočtovom páse (t.j. body kde f = 0) v každom kroku usporiadame vzostupne podľa T. Na čele bude teda hodnota s najmenšou hodnotou T(i, j). Keďže táto hodnota sa už meniť nebude, vyradíme bod (i, j) z pásu, t.j. nastavíme f(i, j) = 2. Zároveň prepočítame hodnoty T(k, l) pre všetkých jeho susedov, kde f(k, l) $\neq$ 2. Ak je f(k, l) = 1, tak zaradíme bod (k, l) do pásu, t.j. nastavíme f(k, l) = 0. Takýmto spôsobom sa pás postupne pohybuje od hranice označenej oblasti smerom dovnútra. Nakoniec sa vypočíta aj posledná hodnota a aproximácia T bude známa vo všetkých bodoch oblasti.

Implementácia Fast Marching metódy v našom programe nie je optimalizovaná, no aj napriek tomu ostáva veľmi rýchla a efektívna. Jej pracovanie je podmienené načítaním obrázku. Po načítaní sa každému pixlu priradí hodnota f = 2 a všade sa nastaví distance funkcia na T = 10^6. V ďalšom kroku užívateľ vyberie miesto, ktoré chce opraviť. Po vyznačení oblasti sa spustí Bresenhamov algoritmus, ktorý pospája body a popri vykresľovaní nastaví hodnoty f = 0 a T = 0. Pôvodné hodnoty, ktoré boli v týchto bodoch sa teda prepíšu. Podobnú prácu následne vykoná scanline algoritmus, ktorý vyplní ohraničenú oblasť a vnútorným bodom priradí hodnotu f = 1. Veľkosť distance funkcie ostáva nezmenená a to T = 10^6. Po skončení scanline algoritmu sa spustí samotná Fast Marching metóda, pričom z riešení kvadratickej rovnice (2) sa vyberá to minimálne.

2.5 Inpainting metóda

Inpainting metóda slúži na získanie novej farby pre poškodený pixel. Metóda získava informácie o farbe pixlov v okolí poškodeného bodu a následne vypočíta novú farbu, ktorou sa pôvodná nahradí. Pixel, ktorý je takto opravený, sa následne pridá medzi už známe body a v ďalšom fungovaní už aj on vplýva na ostatné zatiaľ neopravené body.

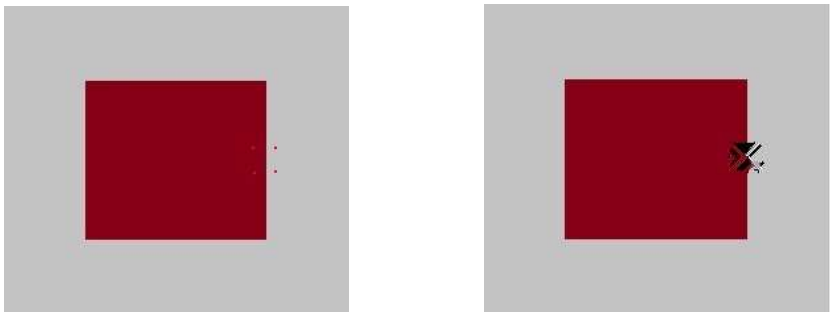
Pri retušovaní postupujeme presne v rovnakom poradí v akom prebiehal výpočet distance funkcie. Začneme od hranice oblasti a jednotlivé body zafarbujeme v smere nárastu distance funkcie T.



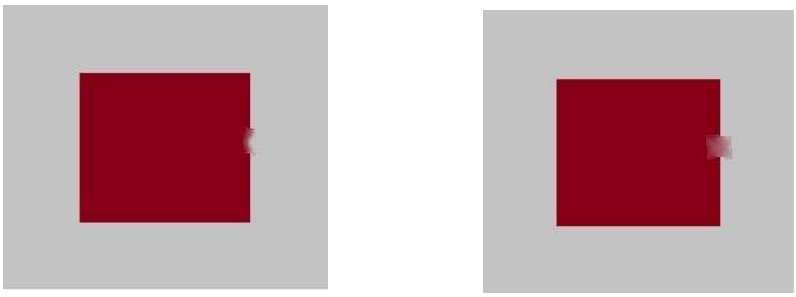
a) *b)*

***Obr. 6**: Princíp Inpainting metódy* [1]

Pri implementácií si ako prvé treba určiť veľkosť okolia retušovaného bodu(p), ktorého body budú určovať novú výslednú farbu p. Veľkosť okolia si označíme ***ε***. Pri väčšej oblasti sa razantne zvyšuje čas potrebný na prebehnutie celej procedúry vzhľadom na to, že narastá množstvo potrebných výpočtov. Avšak ak sa vyberie príliš malá oblasť, nemusí sa dostaviť požadovaný efekt. My sme testovali viacero variantov, pričom hlavný dôraz sa kládol na efekt a až potom na čas. Testy pre jednoduchý obrázok sú znázornené na obr. 7. Pri ***ε*** = 1 metóda prebehla na vyznačenej oblasti zhruba do dvoch sekúnd, no výsledok bol nevyhovujúci (Obr 7b). Vyskytli sa tam aj čierne miesta, ktoré indikujú, že program nebol schopný správne vypočítať novú farbu. Ak bolo ***ε*** zvolené z intervalu <4; 6>, bol výsledok vyhovujúci a čas sa pohyboval okolo piatich sekúnd(Obr 7c). Hodnotu ***ε*** = *5* sme používali ďalej pri všetkých testoch a táto hodnota ostala aj vo finálnej verzii programu. Pri zadávaní omnoho vyšších hodnôt ***ε*** sa viditeľne zvyšoval čas potrebný k fungovaniu programu až na niekoľko minút a takisto sa zhoršoval výsledok (Obr. 7d).



a) *b)*



c) *d)*

***Obr.** 7: Testovanie hodnoty ε:*

a) vyznačená oblasť b) ε = 1 c) ε = 5 d) ε = 35

Podstatou našej retušovacej metódy je to, že farba poškodeného pixlu sa určí ako vážený priemer príspevkov všetkých nepoškodených alebo už opravených pixlov v jeho ε–okolí $B_\varepsilon(p)$. Teraz si povieme, ako sa tento vážený priemer vypočíta. Uvažujme bod p, ktorý treba retušovať a bod q, ktorý sa nachádza medzi nepoškodenými alebo opravenými bodmi v okolí bodu p a teda aj vplýva na tento bod. Príspevok jedného bodu q na farbu bodu p sa určí takto (I je farba bodu)[1]:

$$I(p) = I(q) + \nabla I(q)(p - q)$$

Výsledná farba bodu p sa potom určí ako vážený priemer z príspevkov všetkých bodov množiny $B_\varepsilon(p)$ nasledujúcim spôsobom[1]:

$$I(p) = \frac{\sum_{q \in B\varepsilon(p)} w(p,q)[I(q)+\nabla I(q)(p-q)]}{\sum_{q \in B\varepsilon(p)} w(p,q)} \quad ,$$

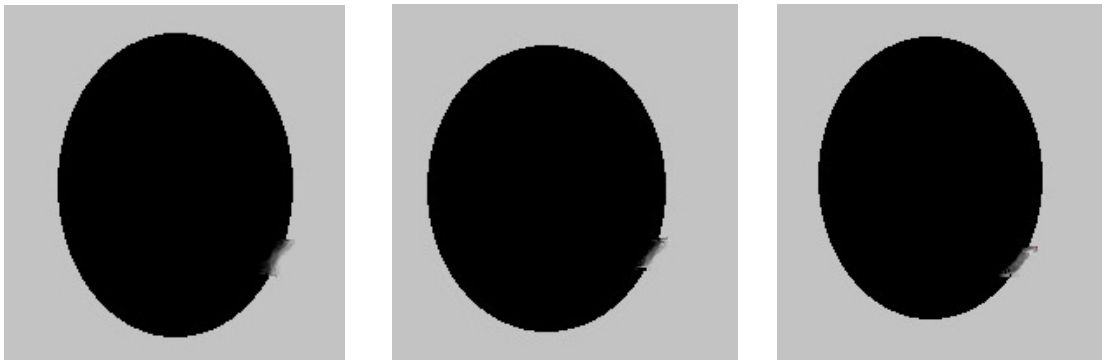
kde ∇I sa aproximuje centrálnymi diferenciami. Teda treba ešte zistiť vplyv(váhu) w(p, q) jednotlivých bodov z daného ε–okolia na práve opravovaný pixel. Veľkosť vplyvu určujú tri hlavné zložky rovnicou[1]:

$$w(p,q) = dir(p,q) * dst(p,q) * lev(p,q) \qquad (2)$$

Smerová zložka (direction component), dir(p, q), zabezpečuje to, že väčšiu váhu budú mať pixely, ktoré sú vzhľadom na bod p umiestnené v smere blízkom smeru gradientu T, t.j. príspevok pixla v smere gradientu. Vypočítame ju rovnicou[1]:

$$dir(p,q) = \frac{p-q}{|p-q|} . N(p),$$

kde N(p) = ∇T sa aproximuje centrálnymi diferenciami. My sme testovali, ako veľmi tento faktor vplýva na výslednú farbu (Obr. 8).



a) *b)* *c)*

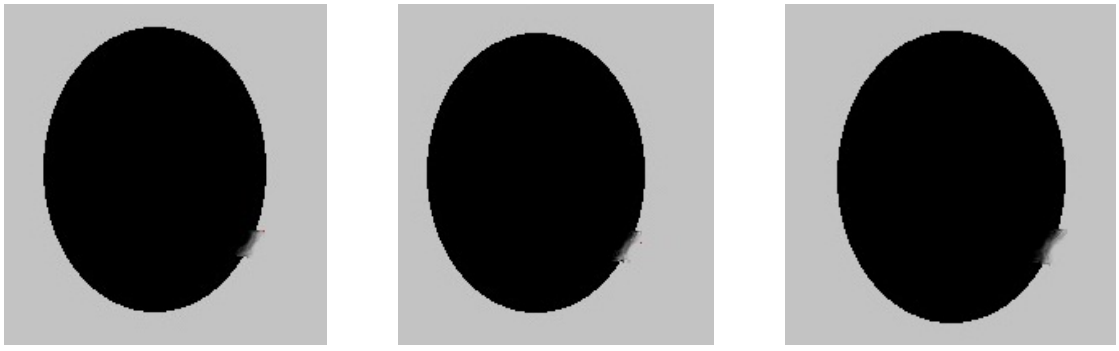
***Obr. 8**: Testovanie vplyvu dir:*

a) použitá pôvodná rovnica (1) b) použitá rovnica (1) s päťnásobnou hodnotou dir c) w = dir

Geometrickú vzdialenosť zahŕňa druhá zložka, rovnicou[1]:

$$dst(p,q) = \frac{d^2}{||p-q||^2},$$

Premenná d je vzdialenosť medzi dvoma susednými pixlami, v našom prípade je vždy d = 1. Keďže sa jedná o zložku vzdialenosti, tak body, ktoré sú vzdialenejšie, vplývajú na výslednú farbu menej.



a) *b)* *c)*

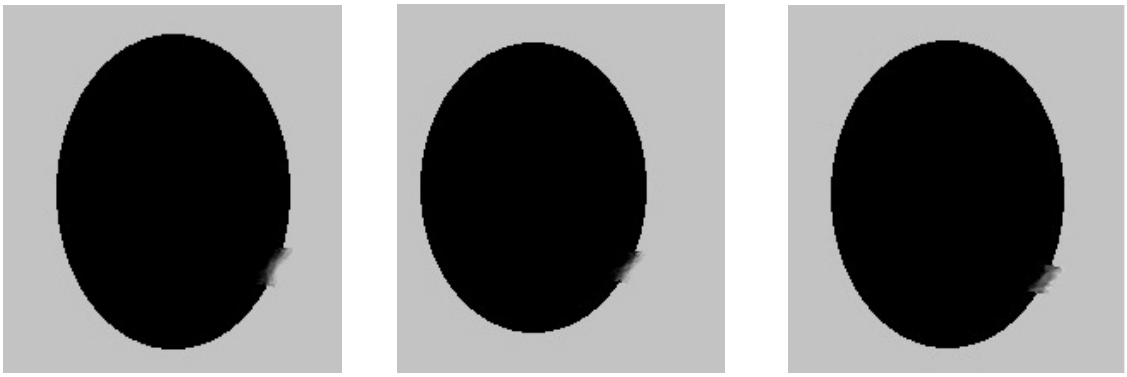
***Obr. 9**: Testovanie vplyvu dst:*

a) použitá pôvodná rovnica (1) b) použitá rovnica (1) s päťnásobnou hodnotou dst c) w = dst

Posledná level set zložka zabezpečuje, že body ležiace na tej istej úrovňovej množine funkcie T ako bod p alebo v jej blízkosti, teda body s rovnakou alebo podobnou hodnotou T, majú väčší vplyv na farbu p ako body vzdialenejšie od úrovňovej množiny T = T(p)[1]:

$$lev(p,q) = \frac{T_0}{1+|T(p)-T(q)|}$$

kde $T_0 = d = 1$.



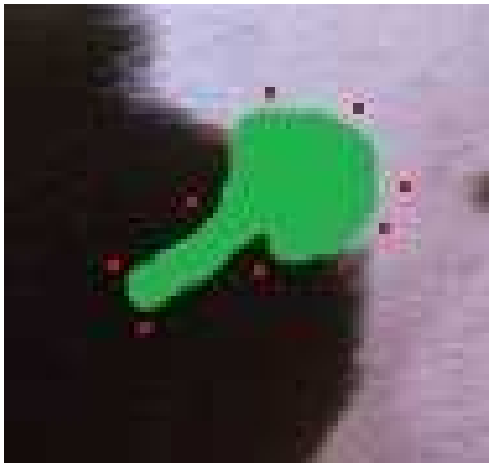
a) *b)* *c)*

***Obr. 9**: Testovanie vplyvu lev:*

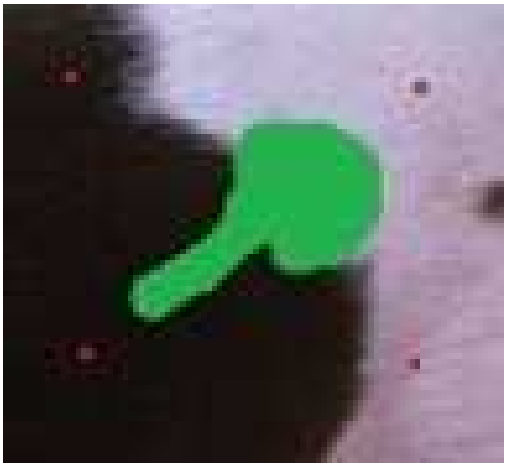
a) použitá pôvodná rovnica (1) b) použitá rovnica (1) s päťnásobnou hodnotou lev c) w = lev

Na základe týchto troch zložiek je teda zrejmé, že výsledok ovplyvňuje aj to, ako vyznačíme poškodenú oblasť.

a)



b)



c)

d)

e)

Obr. 10*: Vplyv vyznačenej oblasti*[11]*:*

a) poškodený obrázok b) vyznačenie poškodenej oblasti opísaním jej tvaru c) vyznačenie poškodenej oblasti bez opísania jej tvaru d) výsledok po vyznačení oblasti z Obr. 10b e) výsledok po vyznačení oblasti z Obr. 10c

Na Obr. 10 sme demonštrovali, aký vplyv na výsledok má t. Kým na Obr. 10d nie je takmer žiadna známka o tom, že by bola fotografia upravovaná, tak na Obr. 10e vznikla machuľa.

3. Implementácia

Keďže sme sa pri implementácií stretli s viacerými problémami, tak sme sa rozhodli ich popísať a zároveň k nim pripojiť aj naše riešenia.

Pri vytváraní softvéru sme potrebovali využívať viacero typov dátových štruktúr, v niektorých prípadoch sme potrebovali dáta zoradiť podľa určitých kritérií. Preto sme sa bližšie oboznámili s knižnicami STL.

Standart Template Library[10], alebo STL, je súbor štandardných knižníc C++, ktoré obsahujú rôzne triedy reprezentujúce rôzne dátové štruktúry a prostriedky na prácu s nimi. STL je súbor generických knižníc, teda takmer každá ich zložka je šablóna.

Pri práci využívame štruktúry set a vector. Sú to v podstate nejaké dynamické polia, ktoré môžu obsahovať ľubovoľný typ dát. Rozdiel medzi nimi je v tom, že vector je neusporiadané pole s rýchlym náhodným prístupom. Naopak set je usporiadané pole, pričom sa zoradí podľa nami definovaného kritéria, a každý prvok sa tu môže vyskytovať iba raz. Náhodný prístup nie je možný, musí sa postupovať vždy od začiatku. Konkrétne u nás bol set využitý napríklad na tabuľku všetkých hrán pri scanline algoritme, kde nám úsečku definovalo viacero hodnôt a podľa týchto hodnôt museli byť aj hrany usporiadané.

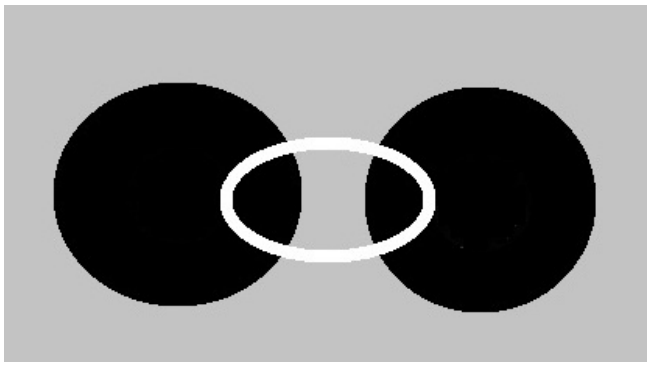
V programe sme využili aj štruktúru multiset, ktorá je obdobou štruktúry set, ale umožňuje duplicitu prvkov. Pomocou nej sme implementovali pás (narrow band) pri výpočte distance funkcie, keďže body v ňom sme chceli mať usporiadané podľa T.

Ďalší problém sa vyskytol pri inpainting metóde. My pracujeme so štandardným RGB systémom, preto bolo nutné ošetriť prípady, kedy by výsledná zložka farby mala byť iná ako z intervalu <0, 255>, ako sa to stalo napríklad na obr. 7b.

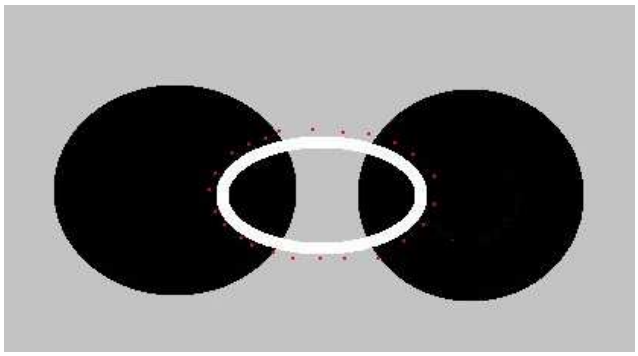
Posledný z problémov stojacich za zmienku bol pri samotnej Fast Marching metóde. Keďže pri výpočtoch potrebujeme hodnoty distance funkcie T v bodoch z ε-okolia retušovaného bodu, musíme výpočet distance funkcie vykonať aj v ε-okolí hranice pôvodnej vyznačenej oblasti smerom von.

4. Pozorovanie a testovanie

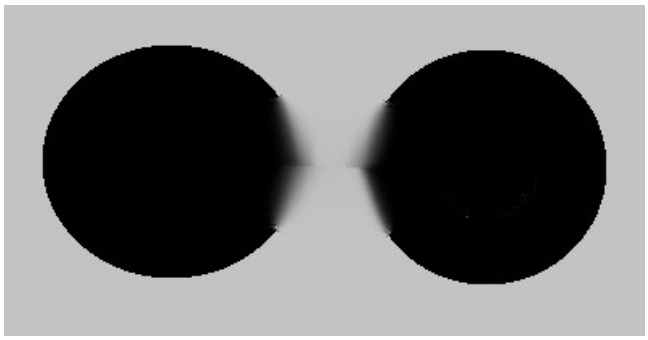
Vzhľadom na to, že ide o automatickú metódu, ktorá vyžaduje minimálnu prácu od užívateľa, tak sa neočakáva, že metóda bude stopercentná. Preto sme hľadali a skúmali jej najlepšie využitie ale aj jej nedostatky.



a)



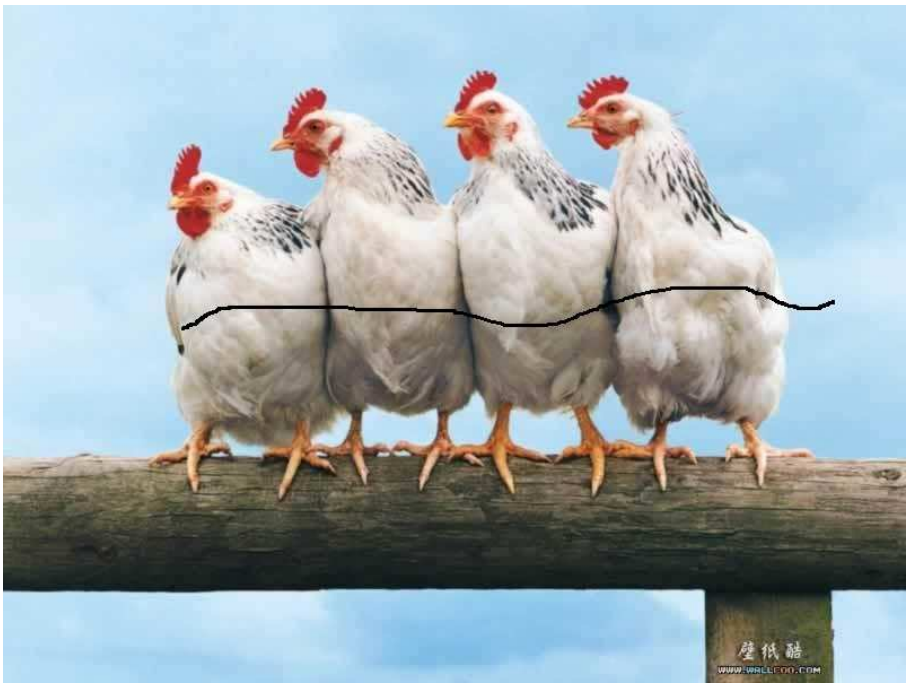
b)



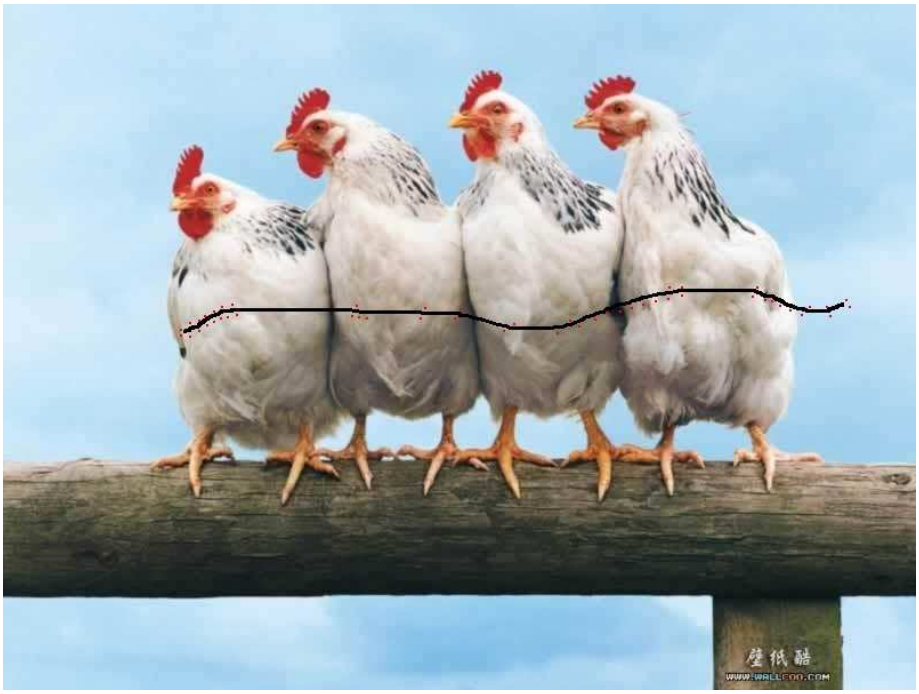
c)

Obr. 11*: Ukážka fungovania metódy*

Na Obr. 11 je vidieť, že síce sme úspešne odstránili bielu čiaru, avšak nedostali sme úplne kruhy, ako sa očakávalo. Toto bol jednoduchý testovací obrázok, ďalšie testy prebiehali na fotografiách a kresbách.



a)



b)

c)

***Obr. 12**: Test metódy na rovnakom pozadí*[12]*:*

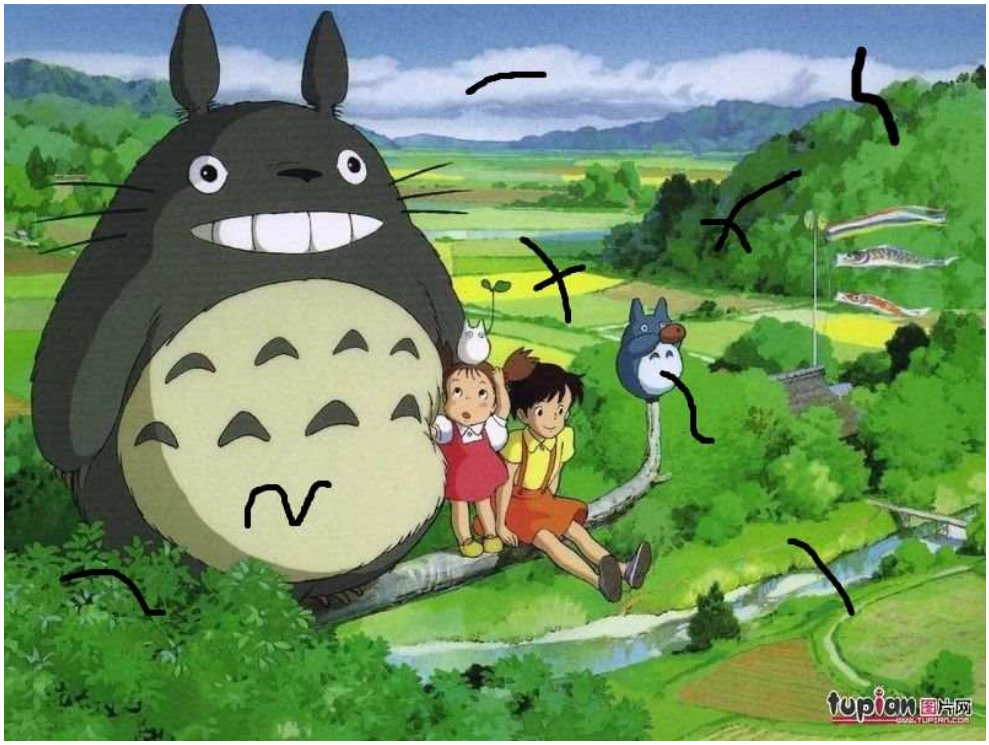
a) poškodený obrázok b) vyznačenie oblasti c) upravený obrázok

a)

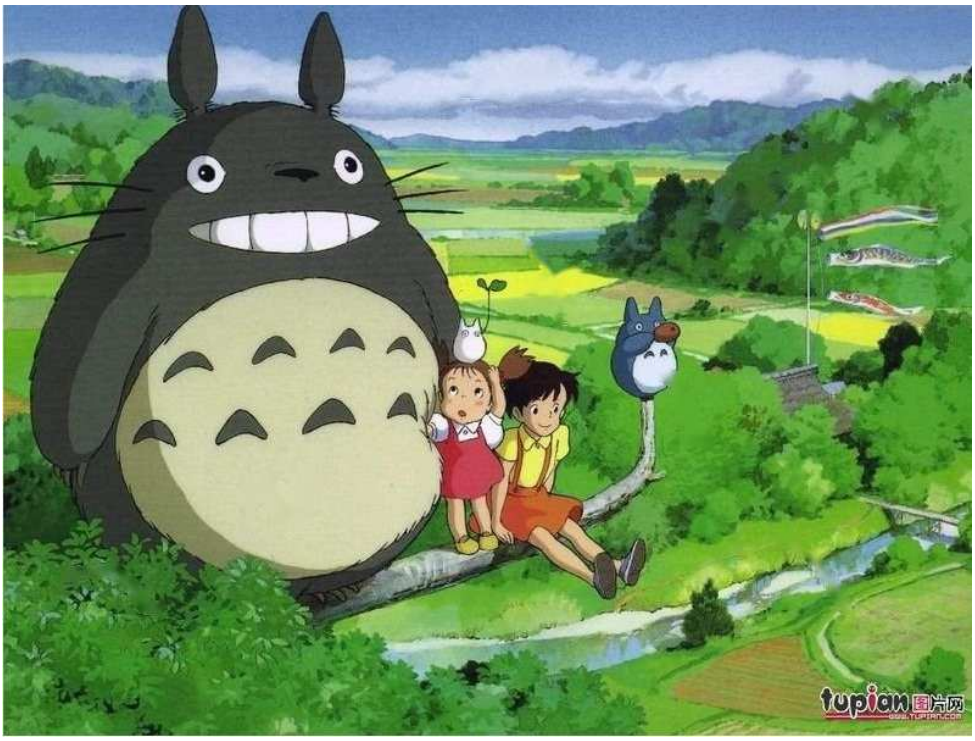
b)

Obr. 13*: Test metódy na viacfarebnom pozadí*[12]*:*

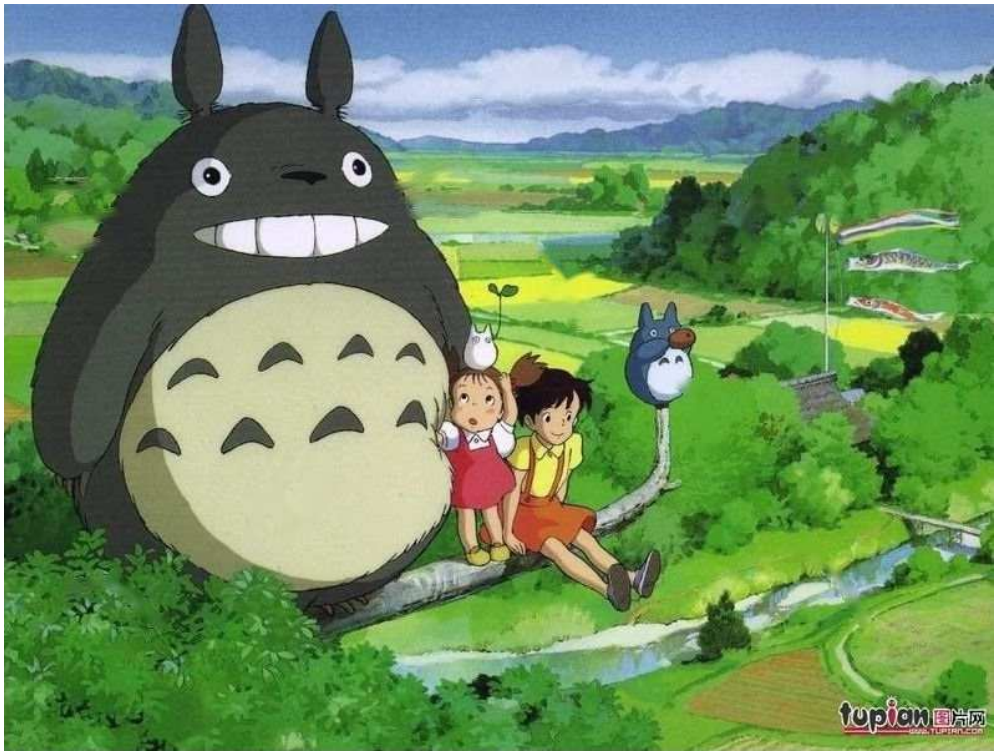
a) poškodený obrázok b) upravený obrázok



a)



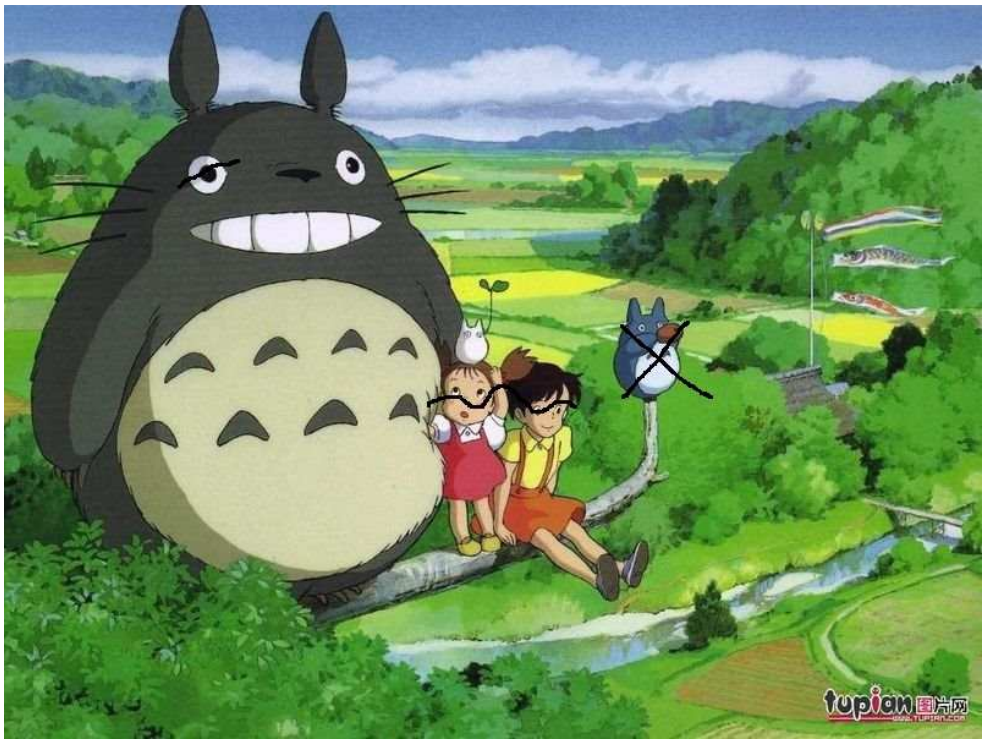
b)



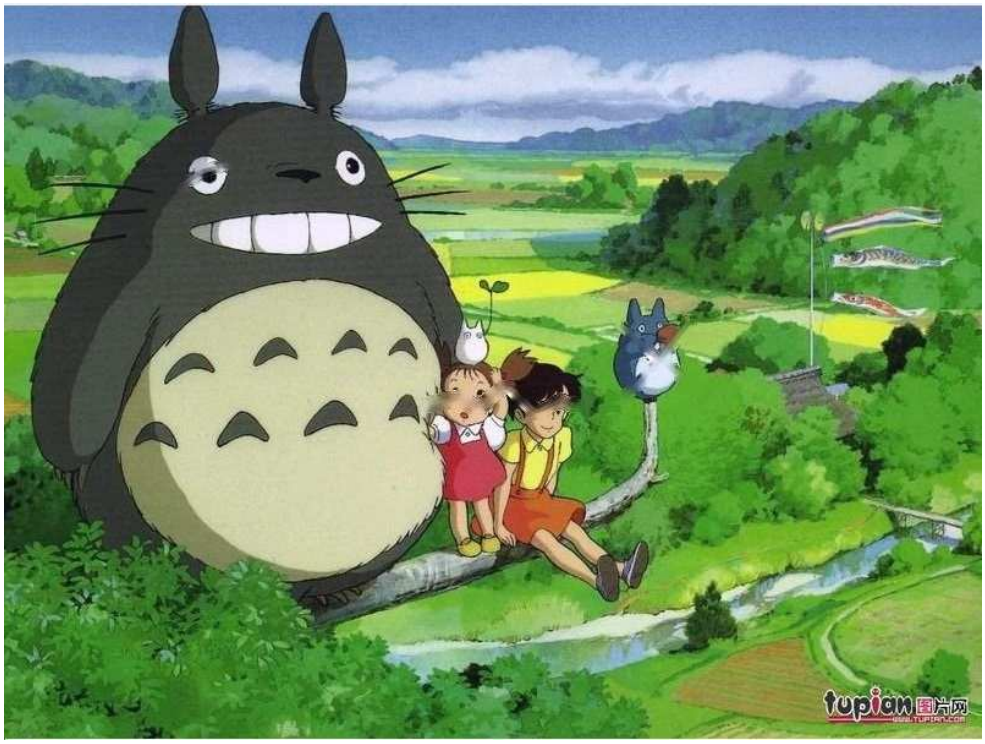
c)

Obr. 14*: Test metódy na pestrom pozadí*[14]*:*

a) poškodený obrázok b) upravený obrázok c) odstránenie fúzov



a)



b)

Obr. 15*: Test metódy na osobách*[14]*:*

a) poškodený obrázok b) upravený obrázok

Najprv sme testovali ako metóda uspeje ak škrabanec ide cez viacmenej jednofarbné územie (Obr. 11). Po odstránení je síce vidno jemnú čiaru, ale vzhľadom na dĺžku a hrúbku poškodenej časti považujeme test za úspešný.

Efekt pre menšiu poškodenú časť na jednofarebnom pozadí sme skúmali v nasledujúcom teste. Avšak tu sme už pridali škrabance aj na rozhranie rôzne zafarbených oblastí ako možno vidieť na obr. 12a. Následne na obr. 12b môžeme pozorovať čiastočný úspech, kedže odstránenie menších škrabancov na bielom pozadí prebehlo bez problémov a dokonca aj na rozhraní dvoch farieb je viditeľný iba minimálny nedostatok. Avšak metóda si už horšie poradila, ak škrabanec išiel cez viacero farieb a najmä cez drobnejšie detaily. Nepovažovali sme to za neúspech, ale testovali sme ďalej, či by sme nenašli využite, aj keď bol zistený tento nedostatok.

Pre náš ďalší test sme si vybrali kreslený, ale hlavne pestrofarebný obrázok. Ak škrabance boli iba na jednofarbných častiach, alebo častiach, kde je krajina ako na obr. 14a, dosiahli sme perfektný výsledok, Obr. 14b, kde je takmer nemožné na prvý pohľad vidieť, že bol obrázok upravovaný. Zistili sme teda, že na členitejších oblastiach sú lepšie výsledky ako na ostrých hranách. Pre lepšiu demonštráciu sme ešte vytvorili obr. 14c.

Posledný test prebiehal na rovnakom obrázku, ale teraz sme sa zamerali na detailnejšie oblasti, ako je oko či tvár, Obr. 15a. Metóda tu mala obrovské problémy a nezvládla jemné obrysy a zanechala viditeľný efekt, Obr. 15b. Je to však aj logické, pretože na dokreslenie takýchto podrobností chýbajú metóde potrebné informácie. Ide o lokálne črty, ktoré sa ťažko získajú ako vážený priemer hodnôt z okolitej oblasti.

5. Záver

V práci sa nám podarilo úspešne implementovať Fast Marching metódu a vytvoriť softvér určený na automatické retušovanie poškodených obrázkov. Program je plne funkčný, má jednoduché a užívateľský príjemné rozhranie.

Ďalej sme v práci skúmali efektivitu a využiteľnosť spomínanej metódy, kde sme zistili, že praktické využite by mohla mať najme pri fotografiách prírody, či iných rozmanitých prostrediach. Jej nedostatky sme našli pri práci s obrázkami, kde sú poškodené drobné dôležité detaily, alebo na rozhraní homogénnych oblastí.

Zoznam použitej literatúry

[1] A. Telea: An An image inpainting technique based on the fast marching method, Journal of Graphics Tools, 9(1), 23-34, 2004

[2] D. Tschumperlé, R. Deriche: Vector-valued image regularization with PDEs: A common framework for different applications, IEEE Transactions on Pattern Analysis and Machine Intelligence, 27(4), 506-517, 2005

[3] E. Rouy, A. Tourin: Visciosity solutions approach to shape-from-shading, SIAM Journal on Numerical Analysis 29(3) (1992), 867-884.

[4] F. Borneman, T. März: Fast image inpainting based on coherence transport, Journal of Mathematical Imaging and Vision, 28(3), 259-278, 2007

[5] J.A. Sethian: 10. Level Set Methods and Fast Marching Methods, Second edition. Cambridge, UK: Cambridge Univ. Press, 1999.

[6] M. Bertalmío, G. Sapiro, V. Caselles, C. Ballester: Image Inpainting, Proceedings of SIGGRAPH 2000, New Orleans, USA, July 2000

[7] M. Remešíková: Rasterizácia, vypĺňanie a orezávanie v 2D, 4-22.

[8] P. Bourgine, P. Frolkovič, K. Mikula, N. Peyriéras, M. Remešíková: Extraction of the intercellular skeleton from 2D microscope images of the early embryogenesis. Lecture Notes in Computer Science 5567 (Proceedings of the 2nd International Conference on scale space and variational Methods in Computer Vision, Voss, Norway), Springer (2009), 38-49.

[9] E. Žára: Moderní počítačová grafika, 2005

[10] Alexander Stepanov Meng Lee: The Standard Template Library. HP Laboratories Technical Report 95-11(R.1), November 14, 1995

[11] http://2.bp.blogspot.com/

[12] http://www.ganoi.com/

[13] http://www.nature.org

[14] http://www.tupian.com