Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Tržaška 25, Ljubljana

Seminarska naloga

Konvolucija za glajenje slik, ostrenje slik in relief

[4]

Grafična tehnika in postopki

Ljubljana, 15. januar 2007

mentor:
dr. Franc Jager

avtor:
Tine Lesjak
63030315

Kazalo

Uvod

Seminarska naloga je namenjena učenju in samostojni raziskavi osnovnih grafičnih postopkov, tehnik in algoritmov za analizo slik (image analysis) ter programskega grafičnega uporabniškega vmesnika (graphical user interface, GUI). Z določenimi prijemi bom pokazal, kako sliko **gladiti, ostriti ali iz nje narediti relief**. Operiral bom z večjo površino slike naenkrat, zato bom govoril o **ploskovnih procesih**.

Za programski jezik sem izbral Javo. **Java** [1] je moderen objektni programski jezik s številnimi že vgrajenimi in neodvisnimi ter velikokrat tudi odprtokodnimi knjižnicami. Njena posebnost je neodvisnost od platforme (operacijskega sistema), v kateri programi tečejo. Java namreč s seboj prinaša vmesno "pogonsko" okolje (runtime environment), med operacijskim sistemom in programom, ki interpretira v vmesno kodo prevedene programe.

Javo sem pred programskim jezikom C/C++ izbral zaradi nekaterih preprostih razlogov:

- v Javi resno programiram že dalj časa,
- je neodvisna od platforme (program se lahko razvija v drugem operacijskem sistemu),
- je sodobna v smislu zasnove (varnost, unicode, izjeme, paketi ...),
- je povsem objektna,
- ima ogromno že priloženih paketov (V/I, RMI, grafika, Swing ...) in
- program je zložen v pakete, kar uredi prevajanje.

Pri programiranju sem uporabljal odlično razvojno okolje (IDE) Eclipse [2], ki je odprtokodno, je privzeto namenjeno programiranju v Javi (out-of-the-box) in uživa precejšen ugled in podporo tudi zaradi velikih imen, kot je npr. IBM.

Čeprav slovenščino zelo spoštujem, sem uporabniški vmesnik oblikoval v **angleščini**. Eden od razlogov je ta, da slovenščina preprosto nima primernega besedišča. Medtem ko se da v angleščini točen pomen kakšne funkcije opisati z eno samo besedo, v slovenščini to dostikrat ni možno. Precej moteče in predvsem nekonsistentno je tudi mešanje jezikov. Če je program oblikovan v drugem jeziku kot operacijski sistem, se lahko zgodi, da so določeni dialogi v programu v jeziku operacijskega sistema, ker uporablja njegove knjižnice.

Metode

Logika

Ploskovni procesi (area processes) so procesi za analizo oz. obdelavo slik, ki uporabljajo večjo površino pri obravnavi ene osnovne enote slike. Bolj natančno: Za ustvarjenje ene točke (pixel) na izhodni sliki uporabi eno točko in nekaj točk okoli nje iz vhodne slike.

Konvolucija

Pri glajenju, ostrenju in drugih efektih se veliko uporablja diskretna konvolucija. Konvolucija (convolution) je le **utežena vsota točk** okoli vhodne točke (točke na vhodni sliki) [3]. Uteži (weights) so predstavljene v obliki matrike, ki ji pravimo konvolucijska maska (convolution mask) ali konvolucijsko jedro (convolution kernel). Dimenzije matrike so ponavadi lihe, tako da je sredina preprosto določljiva.

Pri obdelavi premikamo matriko z masko po sliki. Obravnavana točka je na sredini matrike. Izhodna točka se izračuna tako, da se najprej vsaka vhodna točka, ki se ujema z masko, pomnoži z utežjo, nato pa so dobljeni produkti seštejejo.
Pri tem je potrebno paziti, da izhodne točke shranimo v izhodno sliko, ki ni ista kot vhodna. Če bi izhodno točko shranili v isto sliko, bi pri računanju naslednje izhodne točke dobili napačen rezultat.
Vsota uteži v maski vpliva na intenziteto slike. Če je vsota enaka 1, ima izhodna slika enako srednjo intenziteto kot vhodna. Če želimo intenziteto slike obdržati, moramo sliko **normalizirati**. Normalizacijo (normalization) izvedemo tako, da vsak izhodno točko delimo z vsoto uteži.

$$normFaktor = \sum uteži$$
$$\underline{če}\ (normFaktor == 0),\ \underline{potem}\ normFaktor = 1$$
...
$$izhodnaTočka = \frac{izhodnaTočka}{normFaktor}$$

Enačba 1: Izračun normalizacijskega faktorja in naivna normalizacija vrednosti točke.

Pri takšni izvedbi normalizacije lahko imajo izhodne točke nedovoljene vrednosti (negativne ali prevelike). Negativne vrednosti lahko povzroči tudi maska, ki vsebuje negativne uteži. Zato moramo vsako izhodno točko še omejiti. To lahko storimo na več načinov, verjetno najbolj primeren je ta, da v primeru negativne vrednosti vzamemo kar najmanjšo vrednost točke (to je 0), v primeru prevelike vrednosti pa največjo (to je 255).

$$izhodnaTočka = \min\left\{\left|\frac{izhodnaTočka}{normFaktor}\right|, 255\right\}$$

Enačba 2: Normalizacija in omejitev vrednosti točke.

Robovi slike

Pri realizaciji konvolucije kmalu naletimo na problem, kako obravnavati **robove** slike. Namreč, pri obdelavi točke skrajno levo zgoraj, maska gleda prek zgornjega in levega roba slike. Vprašanje je, kakšne vrednosti vzeti za točke izven robov slike. Možnih rešitev je več:

1. uporabimo vrednost 0,
2. ne obdelamo točk, pri katerih maska gleda izven roba slike - te točke samo prekopiramo,
3. povečamo sliko s kopiranjem roba okoli slike, da dobimo dodaten rob,
4. upoštevamo periodičnost slike - vzamemo točko, ki je na drugi strani slike.

Uporabil sem rešitev št. 2.

Implementacija

Za držanje slike v pomnilniku sem uporabil Javine standardne razrede. Slika je zgrajena iz tako imenovanega **rastra** (raster), ki je polje točk. Kaj predstavlja vrednost točke, podaja **barvni prostor** (color space). V Javi je barvni prostor tipa sRGB [14], kar pomeni, da je vrednost točke predstavljena s kanali v prostoru RGB (red, green, blue). Še več, podpira tudi kanal za prosojnost (alpha channel). Vse skupaj je predstavljeno tako, da ima sivinska slika v eni točki le en kanal, navadna barvna slika tri, prosojna slika pa še četrtega. Vsak kanal ima lahko vrednost od 0 do 255.
Pri konvoluciji je potrebno za vsako točko v sliki posebej upoštevati vsak kanal, kot da bi bil vsak kanal samostojna točka.

Konvolucija je operacija, ki ima v splošnem časovno zahtevnost $O(n^2)$. Ker je pri obdelavi slik ponavadi maska veliko manjša od slike, je časovna zahtevnost enaka O(h*n), pri čemer je h število uteži v maski, n pa število točk v sliki. To velja samo za sivinsko sliko, pri barvni ali prosojni sliki se časovna zahtevnost poveča na O(h*n*c), pri čemer je c število kanalov ene točke.

Ker je konvolucija zelo uporabna operacija in hkrati precej počasna, jo v resnih sistemih izvajamo tako, da signal (v našem primeru sliko) s pomočjo **diskretne Fourierove transformacije** (DFT) pretvorimo v frekvenčni prostor, ga pomnožimo z odzivom (v našem primeru z masko) ter s pomočjo inverzne DFT pretvorimo nazaj v prvotni prostor. Čeprav se zdi, da je ta postopek veliko bolj zahteven, je veliko hitrejši, saj ga lahko izvedemo s FFT (Fast Fourier Transform) algoritmom, ki ima časovno zahtevnost le $O(n*\log_2 n)$.
V Javinem razvijalskem paketu že obstaja implementacija konvolucije za slike (razred java.awt.image.ConvolveOp), ki sem jo preizkusil in je precej hitrejša. Verjetno je realizirana s FFT algoritmom.

Relief

Eden od osnovnih primerov obdelave slike s konvolucijo je izdelava reliefa slike (embossing). Po obdelavi slike je vsak predmet na sliki videti kot **vdolbina ali izboklina na kovinski površini**. Vsota uteži v maski je 0, sredinska utež je 0. Ker ima pri takšni maski veliko točk v sliki zelo nizko ali celo negativno vrednost, se ponavadi vsaki točki prišteje še neka konstanta (npr. kar 128).

$$izhodnaTočka = \min\left\{\left|\frac{izhodnaTočka + 128}{normFaktor}\right|, 255\right\}$$

Enačba 3: Normalizacija in omejitev vrednosti točke s prišteto konstanto.

Ključna lastnost takšne maske je **poudarjanje večjih sprememb vrednosti sosednjih točk** - vidni so samo robovi predmetov. V matematičnem smislu si lahko to predstavljamo kot odvod funkcije, ki za rezultat vrne velike vrednosti tam, kjer je funkcija strma.

Sledijo maske za ustvarjanje reliefa slike. Med seboj se razlikujejo v tem, da so videti "senčene" iz različnih smeri.

$$\begin{matrix} -1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{matrix}$$
Maska 1: Relief.

$$\begin{matrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & -1 \end{matrix}$$
Maska 2: Relief.

$$\begin{matrix} -1 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 1 \end{matrix}$$
Maska 3: Relief.

$$\begin{matrix} 0 & 0 & -1 \\ 0 & 0 & 0 \\ 1 & 0 & 0 \end{matrix}$$
Maska 4: Relief.

$$\begin{matrix} 0 & 0 & 1 \\ 0 & 0 & 0 \\ -1 & 0 & 0 \end{matrix}$$
Maska 5: Relief.

Glajenje

Glajenje (blurring) je proces, pri katerem z rahlo prerazporeditvijo barv (intenzitet) v sliki **zgladimo ostre vrhove in fine detajle**. Proces bi lahko primerjali s filtriranjem z nizkoprepustnim filtrom (low-pass filter). Vsota uteži v maski je enaka 1, pri čemer vsaka utež ni negativna in je manjša od 1. Osnovna maska ima vse uteži enake.

$$\begin{matrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{matrix}$$

Maska 6: Glajenje slik s konstantnimi utežmi.

Iz maske se vidi, da je glajenje le **povprečenje sosednjih točk** - vsaka točka absorbira nekaj sosednje. Ker povprečenje zgladi ekstremne vrednosti točk v sliki, takšna maska znižuje kontrast slike. Zato obstajajo bolj primerne maske, ki so bliže Gaussovemu filtru. Uteži v takšni maski manj vplivajo na srednjo intenziteto slike.

$$\begin{matrix} 1/16 & 1/8 & 1/16 \\ 1/8 & 1/4 & 1/8 \\ 1/16 & 1/8 & 1/16 \end{matrix}$$

Maska 7: Izboljšana maska za glajenje slik.

Ostrenje

Ostrenje (sharpening, crispening) je nasproten proces od glajenja. Ostrenje povečuje razlike med barvami (intenzitetami) sosednjih točk, tako da so **podrobnosti bolj poudarjene**. Primerjali bi ga lahko s filtriranjem z visokoprepustnim filtrom (high-pass filter). Sredinska utež v maski ima pozitivno vrednost, ostale imajo ponavadi negativno.

Spodaj so navadno uporabljane maske za ostrenje slik.

$$\begin{matrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{matrix}$$

Maska 8: Ostrenje.

$$\begin{matrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{matrix}$$

Maska 9: Ostrenje.

$$\begin{matrix} 1 & -2 & 1 \\ -2 & 5 & -2 \\ 1 & -2 & 1 \end{matrix}$$

Maska 10: Ostrenje.

$$\begin{matrix} -1/9 & -1/9 & -1/9 \\ -1/9 & 8/9 & -1/9 \\ -1/9 & -1/9 & -1/9 \end{matrix}$$

Maska 11: Ostrenje.

Ostrenje okrepi kontrast slike in hkrati tudi neželene motnje.

Uporabniški vmesnik

Uporabniški vmesnik skrbi za medsebojno vplivanje uporabnika in računalnika. Uporabnik manipulira, računalnik prikazuje rezultate. Grafični uporabniški vmesnik (graphical user interface, GUI) je poseben primer uporabniškega vmesnika, kjer uporabnik in računalnik medsebojno sodelujeta prek grafičnih elementov in slik, namesto golega besedila. Akcije so izvršene z neposredno manipulacijo grafičnih elementov.
Grafični elementi so podobe (widgets), ki predstavljajo podatke ali akcije na vizualen način. Obstaja jih veliko vrst. Najosnovnejše so okno (window), meni (menu), oznaka (label), gumb (button), ikona (icon), seznam (list) in polje za vnos besedila (text field).

Implementacija

Grafični uporabniški vmesnik programa sem izvedel v Javinem priloženem paketu Swing. Swing skupaj s starejšim paketom AWT vsebuje vse elemente sodobnega grafičnega vmesnika. Swing je platformsko neodvisen in ga je možno uporabljati takoj, brez dodatkov ali posebnih nastavitev sistema (out-of-the-box). Ima naslednje lastnosti [9]:

- vsebuje vse osnovne podobe,
- spremenljiv izgled (look & feel),
- dosegljiv vmesnik za ljudi s posebnimi potrebami (accessibility),
- 2D grafične zmogljivosti za risanje in izpisovanje,
- podporo za povleci in spusti (drag & drop),
- podporo za različne jezike (internationalization).

Izgled vmesnika je ena od zelo pomembnih lastnosti programa. Okoli tega se krešejo mnenja že veliko let. Napredek je ogromen. Vsak izdelovalec platforme bolj ali manj sodobno zasnuje vmesnik, vendar razlike obstajajo. Če dodamo še skladnost s starejšimi sistemi, je jasno, da je platformsko neodvisen vmesnik, ki bi tekel na vseh sistemih in po vrhu še enako izgledal, praktično nemogoče narediti. No, skoraj, saj je Swing prav to.
Ker Swing podpira **spreminjanje izgleda** na zelo enostaven način, sem uporabil dodaten paket JGoodies Looks. Ta paket je med razvijalci zelo znan, saj se lahko pohvali z izboljšano berljivostjo, estetiko, odlično konsistentnostjo in izredno natančnim oblikovanjem [10]. Sam izgled pa ne pomaga veliko, če so podobe neustrezno poravnane oz. kar razmetane. **Poravnava** je eden od zelo perečih problemov, saj so podobe med seboj precej različne v velikosti, vsebujejo različno velike podatke, so lahko hierarhično postavljene ter omogočajo spreminjanje velikosti. Težave se da omiliti, če podobe skušamo postaviti tako kot na obrazcu (form) in pri tem uporabimo dobro orodje. Eno najboljših, ki sem ga tudi uporabil, se imenuje JGoodies Forms.
Oba in edina dodatna paketa, ki sem ju uporabil, JGoodies Looks in Forms sta brezplačna, napisana v Javi in sta na voljo kot knjižnici. Avtor Karsten Lentzsch ponuja na svoji spletni strani http://www.jgoodies.com še druge zanimive pakete.

Načela za načrtovanje

Razvoj grafičnih uporabniških vmesnikov je pester in ni prišel kar čez noč. Razvijalci so skušali koncepte za načrtovanje GUI-jev kar se da formalizirati, nekako dognati njihovo priljubljenost in po možnosti še izboljšati. Na osnovi človekovega mentalnega modela in njegovih fizičnih ter psiholoških lastnosti so nastala načela za načrtovanje GUI-jev.

Poglejmo si, kako sem jih upošteval v treh točkah:

1. Uporabniku zagotovi nadzor

Omogoči uporabo tipkovnice in/ali miške (fleksibilnost)	Omogoča že Swing, dodal sem mnemonike k vsem gumbom in menijem.
Omogoči prekinitev dolgih opravil (prekinljivost)	Zahtevna naloga, ker moramo manipulirati z nitmi. Program je preenostaven, zato prekinljivosti nisem omogočil.
Prikazuj obvestila in namige (pomoč)	Napake pri odpiranju in shranjevanju slike; namigi (tooltips) se prikažejo pri vseh pomembnih podobah.
Zagotovi takojšnje in ponovljive akcije ter povratno informacijo (odzivnost)	Program je enostaven, zato se akcije izvajajo hitro.
Zagotovi značilne poti in izhod (navigacija)	Program ima glavni meni ter osrednjo nadzorno ploščo, ki vsebuje značilne funkcije. Nima orodne vrstice; ima statusno vrstico.
Prilagodi se uporabnikom z različnimi nivoji znanja (dostopnost)	Program nima naprednih funkcij, zato ne potrebuje naprednega načina.
Zagotovi jasnost in čistost vmesnika	To zagotavlja pravilna uporaba JGoodies Looks

(preglednost)	in Forms.
Omogoči spreminjanje lastnosti vmesnika (želje)	Program nima posebnih lastnosti. Konvolucijo je mogoče dodatno nastaviti; masko je mogoče nastaviti na prednastavljeno matriko.
Omogoči neposredno manipulacijo z grafičnim vmesnikom (interaktivnost)	Omogoča Swing.
Uporabljaj funkcije pametno (nedvoumnost)	Program blokira določene funkcije, ko niso na voljo.

2. Zmanjšaj obremenitev uporabnikovega spomina

Razbremenjuj kratkotrajni spomin (pomnjenje)	Funkcije izreži/kopiraj/prilepi so omogočene pri spreminjanju maske samo prek tipkovnice.
Zanašaj se na razpoznavanje in ne na spomin (prepoznavanje)	Vsi gumbi in meniji so na kratko opisani in zloženi v pravilnem vrstnem redu. Maska je oblikovana v matriko.
Zagotovi vizualne namige (informiranost)	Program nima pomoči v smislu vodnika. Namigi so prisotni.
Zagotovi razveljavitev in ponovitev akcij (preprostost)	Masko je mogoče nastaviti s prednastavljenimi vrednostmi; izvorno sliko je mogoče ponastaviti na originalno.
Zagotovi bližnjice (hitrost)	Vsi gumbi in meniji imajo mnemonike, nekatere funkcije imajo tudi določene pospeševalnike; glavne funkcije so na osrednji nadzorni plošči.
Podpiraj način gradnik - akcija (intuitivnost)	Omogoča vsak GUI.
Uporabljaj podobnosti iz realnega sveta (prenos)	Program nima ikon. Maska je postavljena v obliki matrike.
Uporabljal progresivni dostop (vodljivost)	Program je preenostaven in nima naprednih možnosti.
Podpiraj vizualno čistost (organiziranost)	Funkcije so smiselno grupirane.

3. Zagotovi konsistentnost vmesnika

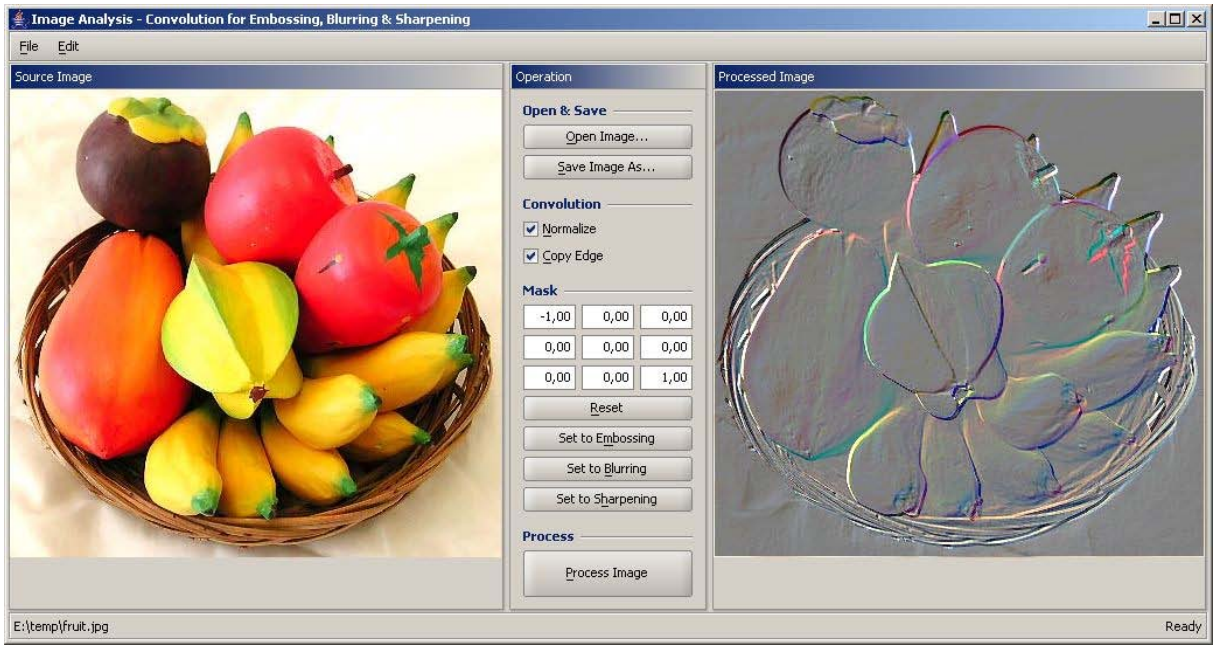
Ohranjaj kontekst uporabnikovih opravil (zveznost)	Program je enostaven in ima skladen slog interakcije.
Ohranjaj enovitost v predstavitvi informacij, obnašanju gradnikov in tehnikah interakcije (izkušnje)	Podprto iz Swinga, dodatno izboljšano z JGoodies Looks.
Ohranjaj enovitost rezultatov (pričakovanje)	Omogočajo podobe Swinga.
Zagotovi estetsko privlačnost in polnost (izgled)	Zagotovljeno iz Swinga in JGoodies Looksa.
Vzpodbujaj preiskovanje (napovedljivost)	Program je prijazen: dialoga za odpiranje in shranjevanje slike se ne zapreta, dokler ni namen dosežen.

Meniji v programu se zgledujejo po modelu FEVH. Meniji, narejeni po tem modelu, niso objektno usmerjeni, temveč podajajo statične funkcije vidne v celotnem programu. FEVH meniji vsebujejo vsaj naslednje menije: Datoteka (File), Urejanje (Edit), Pogled (View), Pomoč (Help). Zadnjih dveh v svojem programu nisem uporabil. Meni Pogled se uporablja za vključitev ali izključitev posameznih funkcij, menijev, oken ali orodne in statusne vrstice. Menija Pomoč program ne vsebuje, ker zanj nisem spisal pomoči.

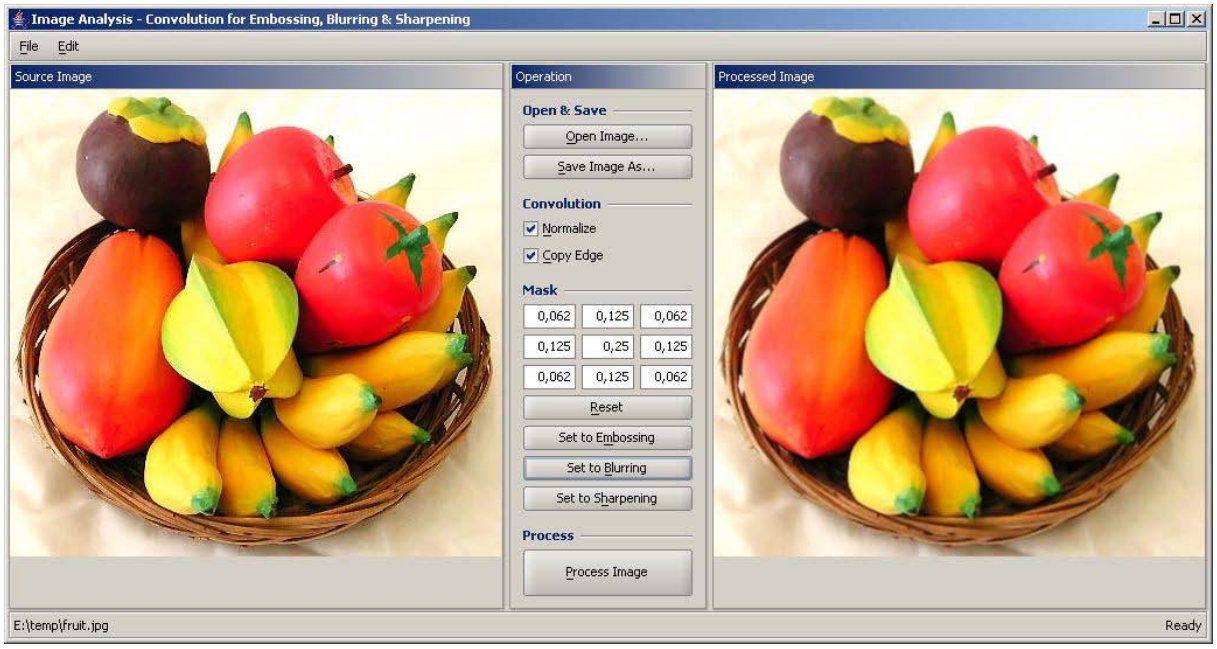
WOSH model menijev je za razliko od FEVH objektno usmerjen. Meniji se prilagajajo glede na trenutno aktiven (fokusiran) gradnik.

Rezultati

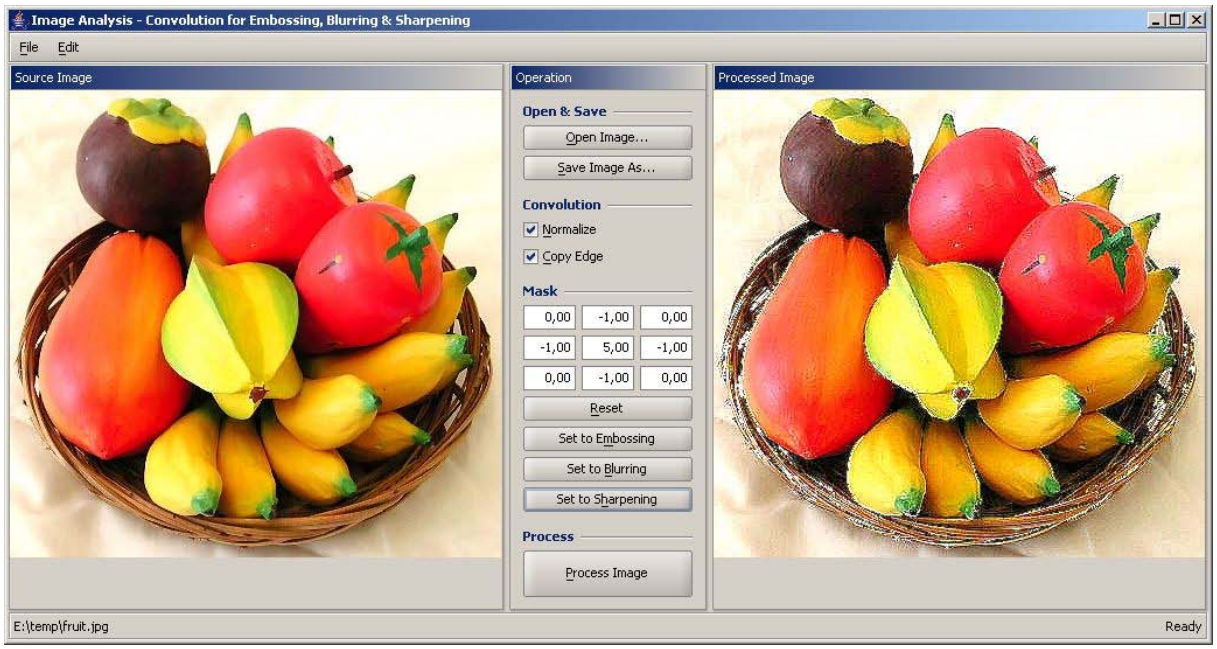
Konvolucija



Slika 1: Na levi strani je originalna slika, na desni je njen relief. Uporabljena je bila maska št. 1.



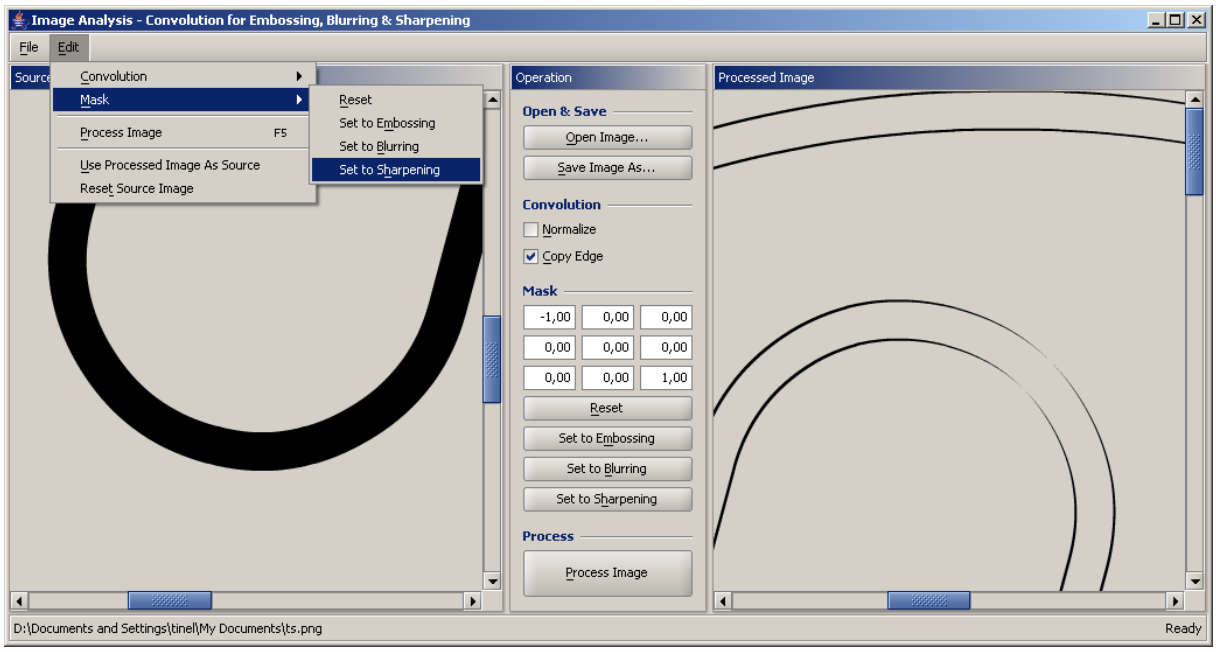
Slika 2: Zglajena slika z masko št. 7.



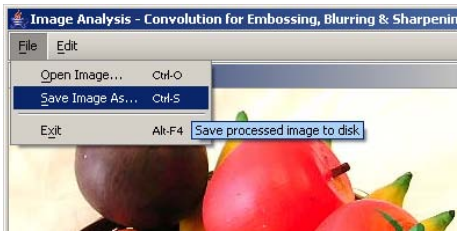
Slika 3: Izostrena slika z uporabo maske št. 8.

Uporabniški vmesnik

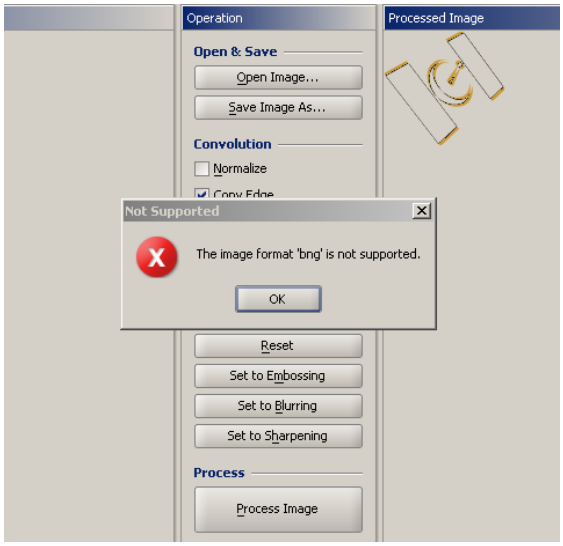
Po zagonu programa imamo pred seboj okenski vmesnik, ki vsebuje menije, statusno vrstico, levo notranje okno za izvorno sliko (Source Image), na sredini osrednjo nadzorno ploščo (Operation) in desno notranje okno za obdelano sliko (Processed Image). Najprej naložimo (Open Image...) sliko, ki jo želimo obdelati. Slika se prikaže v levem notranjem oknu. Nato nastavimo ustrezno masko ročno ali s prednastavljenimi vrednostmi (Reset, Set to Embossing, Set to Blurring, Set to Sharpening), ter sliko obdelamo (Process Image). Obdelana slika se prikaže desno. Obdelano sliko je možno sedaj uporabiti kot izvorno (Use Processed Image As Source). Če to storimo, lahko izvorno sliko tudi ponastavimo (Reset Source Image). Obdelano sliko lahko tudi shranimo (Save Image As...). Nazadnje program še končamo (Exit).
Statusna vrstica levo prikazuje pot do trenutno odprte (originalne) slike, na desni pa stanje programa. Kadar program izvaja daljšo operacijo na zelo veliki sliki, piše "Opening image...", "Saving image..." ali "Processing image...", ustrezne funkcije so blokirane, miškin kazalec pa kaže peščeno uro.
Kadar operiramo z dimenzijsko veliko sliko, se v levem in desnem notranjem oknu pojavijo drsniki, tako da je mogoč ogled celotne slike.



Slika 4: Izgled vmesnika programa. Vidna je zgradba menijev in osrednje nadzorne plošče. Vidni so tudi drsniki in rezultat obdelave prosojne PNG slike.



Slika 5: File meni. Vidni so standardni mnemoniki in pospeševalniki ter namig.



Slika 6: Obvestilo o napaki pri shranjevanju slike v nepoznanem formatu.

Razprava

Rad bi potrdil svojo pravilno odločitev, da sem program ustvaril v programskem jeziku Java in razvijalskem okolju Eclipse. Nekaj razlogov sem opisal že v uvodu. Dodaten razlog je učinkovitost, saj sem za izgradnjo grafičnega vmesnika potreboval razmeroma malo časa. Nič časa pa nisem izgubil za prevajanje programa ali nastavljanje pravilnih parametrov, saj sem imel orodje že nameščeno in pripravljeno.
Kar nekaj mojih sošolcev se je pritoževalo nad programskim jezikom C/C++ in knjižnicama za izgradnjo GUI-ja Motif ali GTK+. Nekaj jih seminarske naloge tudi ne bo oddalo. Po pogovoru z njimi, sem ugotovil, da so razlogi najverjetneje naslednji:

- niso uporabili nobenega razvijalskega orodja,
- niso razvijali programa v svojem "domačem" operacijskem sistemu,
- C/C++ in Motif/GTK+ so ločeni projekti z ločenimi dokumentacijami, različnimi napotki, parametri, formati kode itd.

Nekaj težav sem imel pri izgradnji algoritma za konvolucijo. Težave so bile na moji strani, saj sem, tako kot ponavadi, prehitro hotel narediti vse in pri tem pozabil prebrati osnovno literaturo. Slika je namreč bila po obdelavi z masko, ki nima vsote uteži enake 0 ali 1, nenormalno svetla ali temna, kar je posledica povečanja ali zmanjšanja srednje intenzitete slike. Težava pa ni bila tako enostavna, saj bi bilo morebitno napako v samem algoritmu precej težko najti. Napaka ni bila v algoritmu, rešiti se jo da enostavno in standardno z normalizacijo intenzitete (barv).
Nekaj zapletov sem pričakoval tudi pri slikah z dodatnim barvnim kanalom oz. prosojnim kanalom - slike formata PNG. Na srečo je Java za to že poskrbela in zna v osnovi delati tudi z delno prosojnimi slikami, še več, kanalov lahko imamo poljubno število! Prosojni kanal je implementiran tako, kot da bi slika imela še četrto barvno komponento. Časovna zahtevnost konvolucije je pri takšnih slikah zaradi tega nekoliko višja (O(h*n*4)), a je algoritem tako napisan, da pri sivinskih slikah s samo enim kanalom obdeluje samo en kanal (O(h*n)).

Če bi se s podobnimi opravili ukvarjal za komercialno in ne akademsko rabo, bi uporabil naprednejše algoritme, ki so seveda že na napisani. Nekaj takšnih pride skupaj z Javinim razvijalskim paketom. Za konvolucijo lahko uporabimo razred java.awt.image.ConvolveOp. Težava le-tega je, da je sam algoritem konvolucije zaprt s strani Suna (zanj ni izvorne kode), kar onemogoča nadaljnje fine optimizacije ali dodajanje efektov. Zanimivo je tudi to, da sem pri uporabi tega razreda dobil rahlo drugačne rezultate pri vseh treh ploskovnih procesih.
Mimogrede sem na spletu naletel tudi na precejšnjo bazo filtrov JH Labs [12], napisanih v Javi, ki so prosto dostopni in jih je enostavno uporabiti. Filtrov je precej vrst: za barve, za spreminjanje oblike, za posebne efekte, za teksture itd. Naj omenim, da imajo na spletni strani opisanih kar 20 filtrov samo za glajenje in ostrenje. Izgledajo res profesionalno.

Še vedno je vse preveč dela z grafičnim uporabniškim vmesnikom. Tukaj mislim na "umazano" delo, ko se je potrebno ukvarjati s stvarmi, ki nimajo veliko skupnega s tem segmentom. Gradnja uporabniškega vmesnika bi morala biti na višjem nivoju od samega programskega jezika, tako da bi se lahko programer-oblikovalec osredotočil na primarni cilj. Standardne stvari, kot so internacionalizacija, mnemoniki, pospeševalniki, dodajanje odzivnih funkcij, skupne/večkrat rabljene funkcije, pomoč, upravnik oken, prikaz in podpora spletne vsebine, tiskanje itd., bi morale biti zapakirane in enostavno uporabne, da jih ne bi bilo potrebno vsakokrat vnovič pisati. Za lažjo predstavo: vse te funkcije bi se moralo dati napisati kar v XML dokumentu!
Tega si seveda nisem sam in edini izmislil, kar pomeni, da takšni izdelki že obstajajo. Reče se jim platforma za bogatega odjemalca (rich client platform, RCP) in so trenutno nekje vmes, med osnovnim programskim jezikom in XML dokumentom. Tipični predstavniki te platforme v Javi so Eclipse RCP, Spring RCP in NetBeans [15]. Ti že vsebujejo veliko večino bogatejših grafičnih podob in pametnejših funkcij. Notranja okna (views), na primer, urejamo kar z imeni razreda v XML dokumentu, prav tako odzivne funkcije. Obstajajo pa tudi že višji programski jeziki, že dolgo se šušlja o XAML-u [13], ki v XML obliki opiše objekte (gradnike) in njihove lastnosti. Za grafični vmesnik tako potrebujemo samo XAML dokument in ustrezno platformo (.NET 3.0). A je potrebno povedati, da je XAML trenutno še bolj na nivoju spletne animacije v stilu SVG-ja ali Flasha in ne kot okenska aplikacija. Vendar je ideja odlična.

Viri

[1] Java Technology: The Source for Java Developers. Sun Microsystems, Inc., 9. januar 2007. http://java.sun.com

[2] Eclipse.org home: Eclipse - an open development platform. The Eclipse Foundation, 9. januar 2007. http://www.eclipse.org/

[3] Crane R. A simplified approach to image processing: classical and modern techniques in C. Prentice Hall, Inc., 1997.

[4] Jager F. Grafična tehnika in postopki: Predloge za predavanja. 2006.

[5] Java 2 Platform, Standard Edition, v 1.4.2: API Specification. Sun Microsystems, Inc., 10. januar 2007. http://java.sun.com/j2se/1.4.2/docs/api/

[6] Wikipedia contributors. Graphical user interface. Wikipedia, The Free Encyclopedia, 11. januar 2007, 16:53. http://en.wikipedia.org/w/index.php?title=Graphical_user_interface&oldid=100017658

[7] Wikipedia contributors. User interface. Wikipedia, The Free Encyclopedia, 8. januar 2007, 19:54. http://en.wikipedia.org/w/index.php?title=User_interface&oldid=99384974

[8] Wikipedia contributors. Widget (computing). Wikipedia, The Free Encyclopedia, 5. januar 2007, 15:37, http://en.wikipedia.org/w/index.php?title=Widget_%28computing%29&oldid=98657077

[9] The Java Tutorials: About the JFC and Swing. Sun Microsystems, Inc., 13. januar 2007. http://java.sun.com/docs/books/tutorial/uiswing/start/about.html

[10] JGoodies: Looks. JGoodies. 13. januar 2007. http://www.jgoodies.com/freeware/looks/index.html

[11] JGoodies: Forms. JGoodies. 13. januar 2007. http://www.jgoodies.com/freeware/forms/index.html

[12] Jerry. Java Image Processing: Java Image Filters. JH Labs. 15. januar 2007. http://www.jhlabs.com/ip/filters/index.html

[13] XAML: Your XAML Information Page. Mobiform Software Inc., 15. januar 2007. http://www.xaml.net/

[14] Stokes M., Anderson M., Chandrasekar S., Motta R. A Standard Default Color Space for the Internet - sRGB. W3C. 5. november 1996. http://www.w3.org/Graphics/Color/sRGB.html

[15] Wikipedia contributors. Rich Client Platform. Wikipedia, The Free Encyclopedia, 14. januar 2007, 00:18. http://en.wikipedia.org/w/index.php?title=Rich_Client_Platform&oldid=100550575