UNIVERZITET U BEOGRADU

ELEKTROTEHNIČKI FAKULTET

****

**PRIKAZ HISTOGRAMA, KARAKTERISTIKA I PERFORMANSI FILTERA NA ANDROID OPERATIVNOM SISTEMU**

Mentor: Studenti:

dr Irini Reljin, vanr. prof. Sofija Purić, dipl. ing.

Milan Mladenović, dipl. ing.

Beograd, jul 2014.

Sadržaj

[1. Uvod 3](#_Toc391927529)

[2. Postavka problema 3](#_Toc391927530)

[3. Karakteristike obrađenih filtera 3](#_Toc391927531)

[3.1. Primena unarnih filtera nad slikom 5](#_Toc391927532)

[3.1.1 Invert filter 6](#_Toc391927533)

[3.1.2 Black&White filter 8](#_Toc391927534)

[3.1.3 Brightness 10](#_Toc391927535)

[3.1.4 Contrast 12](#_Toc391927536)

[3.1.5 Flip vertical 13](#_Toc391927537)

[3.1.6 Flip horizontal 14](#_Toc391927538)

[3.1.7 Grayscale 15](#_Toc391927539)

[3.1.8 Gama korekcija 16](#_Toc391927540)

[3.1.9 Color filter 19](#_Toc391927541)

[3.1.10 Shading filter 21](#_Toc391927542)

[3.2. Primena binarnih filtera nad slikama 23](#_Toc391927543)

[3.3. Primena konvolucionih filtera nad slikom 23](#_Toc391927544)

[4. Primena histograma 23](#_Toc391927545)

[5. Pregled performansi filtera 23](#_Toc391927546)

[6. Zaključak 23](#_Toc391927547)

[7. Literatura 23](#_Toc391927548)

# Uvod

# Postavka problema

# Karakteristike obrađenih filtera

Primena filtera nad slikom predstavlja proces gde se nad informacijom koju nosi piksel, dakle *RGB* vrednosti, primenjuje matematička funkcija da bi se proizvela odgovarajuća alternativna verzija slike. Dakle, filteri predstavljaju programske operacije koje nam omogućuju primenu različitih efekata nad slikama.

Slika je funkcija  za neko N i M iz skupa prirodnih brojeva, pri čemu su N i M dimenzije slike u pikselima. Svaki piksel predstavljen je sa tri komponente: crvena (*R*), zelena (*G*) i plava (*B*), i to u opsegu od *0...255* za svaku komponentu. Neka je skup slika označen sa *I.* Filter predstavlja funkciju  koja preslikava postojeću sliku u novu sliku uz promenu (*RGB*) vrednosti svakog piksela po odgovarajućem algoritmu. Ako je skup svih filtera označen sa *F,* onda simbol *f(I)* predstavlja primenu nekog od filtera iz skupa *F* nad nekom od slika iz skupa slika *I.*

U ovom poglavlju biće dat pregled svih obrađenih filtera nad izabranom slikom, koji su podeljeni u tri kategorije: unarni, binarni i konvolucioni filteri. Biće opisan način rada samog filtera, kao i način na koji je filter implementiran u aplikaciji uz prikaz delova koda. Osim toga biće prikazan i rad tih filtera uz slikovit prikaz delova aplikacije.

## Primena unarnih filtera nad slikom

Pod pojmom unarnih filtera podrazumevamo one koji kao ulazni parametar zahtevaju samo jednu sliku nad kojom se primenjuje taj filter i kao rezultat vraćaju novu sliku nakon primene tog filtera. Pored toga unarni filteri mogu imati i argumente koji mogu predstavljati intenzitet primene ili jačinu neke od *RGB* komponenti koje su potrebne za primenu tog filtera nad slikom.

Filteri koji će biti opisani u nastavku rada su sledeći: invert, black&white, brightness, contrast, flip vertical, flip horizontal, grayscale, gamma correction, color filter, saturation, hue i shading filter.

### Invert filter

Invert filter, poznat još pod imenom negativ, filter je koji menja originalnu vrednost piksela u njegovu inverznu vrednost, odnosno invertuje vrednost svakog piksela. Svaki piksel na slici čuva informacije o četiri kanala: alpha, red, green i blue. Međutim, alpha ne odražava vizuelno boju slike, tako da se inverzija primenjuje na ostala tri kanala, po formuli:

*nova\_vrednost* (*R, G, B*) = 0xFF – *trenutna\_vrednost*(*R, G, B*) (3.1)

Najvažniji deo koda koji obavlja invertovanje piksela prikazan je u tabeli 3.1:

Tabela 3.1: Invert filter nad slikom

**for** (**int** x=0; x<width; x++) {

**for** (**int** y=0; y<height; y++) {

r=255-Color.*red*(colorArray[y\*width+x]);

g=255-Color.*green*(colorArray[y\*width+x]);

b=255-Color.*blue*(colorArray[y\*width+x]);

colorArray[y\*width+x]=Color.*rgb*(r,g,b);

returnBitmap.setPixel(x,y,colorArray[y\*width+x]);

}

}

Pri tome, width označava širinu slike, height visinu, a colorArray je niz celobrojnih vrednosti u kome pamtimo vrednosti piksela (ima width\*height elemenata). Vrednosti komponenata piksela (alpha, red, green, blue) su uskladištene u vrednost piksela na Android operativnom sistemu na sledeći način:

*pixel* = (*alpha* << 24) | (*red* << 16) | (*green* << 8) | *blue* (3.2)

Metode Color.red, Color.green i Color.blue vraćaju crvenu, zelenu i plavu komponentu odgovarajuće vrednosti piksela koja se prosleđuje kao argument metode, respektivno. Zbog poznatog načina skladištenja pojedinačnih komponenti piksela, te vrednosti smo mogli dobiti i na sledeći način:

*red* = (*color* >> 16) & 0xFF (3.3)

*green* = (*color* >> 8) & 0xFF (3.4)

*blue* = *color* & 0xFF (3.5)

Metoda Color.rgb(r, g, b) vraća celobrojnu vrednost piksela na osnovu vrednosti pojedinačnih komponenti koje se prosleđuju kao argumenti poziva metode, a koje treba da budu u opsegu [0..255]. Vrednost alpha komponente je implicitno 255, što označava potpunu neprozračnost (neprovidnost). Metoda returnBitmap.setPixel(x, y, colorArray[y\*width+x]) upisuje odgovarajuću vrednost piksela na odgovarajuće mesto u slici (na specificiranu x i y koordinatu piksela). Slika je na Android operativnom sistemu predstavljena preko objekta Bitmap.

Na slikama 3.1 i 3.2 prikazane su originalna slika i slika nakon primene invert filtera:



3.1 Originalna slika pre invert filtera

3.2 Slika nakon invert filtera

### Black&White filter

Black and white predstavlja filter koji za postojeću sliku koja se zadaje, kao i *scale* parametar koji predstavlja intenzitet, vraća sliku čiji pikseli mogu imati samo crnu ili belu boju. Tj vrednosti komponenti *RGB* za sve piksele mogu biti ili *0,0,0* ili *255,255,255*. Filter prihvata sliku kao prvi argument, i parametar *scale* koji predstavlja realan broj od *0* do *100.* Filter zatim pravi lokalnu kopiju prosleđene slike, i računa prag na osnovu *scale* parametra po formuli:



Zatim se vrši obilazak matrice i za svaki piksel određuje se njegovo osvetljenje po formuli.



Pri čemu je *x[i][j]* piksel na poziciji *i,j,* a anotacija *.red, .green, .blue,* označava da se uzima crvena (*R*), zelena (*G*), plava (*B*) komponenta boje tog piksela. Nakon toga se na osnovu osvetljenosti piksela i praga računa njegova nova vrednost i to ako je *newValue > t* uzima se 255, ako je *newValue < t* uzima se *0*, ako je *t > 127.5* uzima se *255*, ako je *t < 127.5* uzima se *0* i na kraju ako nije ni jedan od prethodnih uslova uzima se *255.* Zatim se nova vrednost postavlja kao parametar za sve tri komponente (*RGB*) piksela.

scale = (scale/100)\*255;

for(int i = 0; i < width; i++) {

for(int j = 0; j < height; j++) { r = Color.red(colorArray[j \* width + i]);

g = Color.green(colorArray[j \* width + i]);

b = Color.blue(colorArray[j \* width + i]);

newValue=0.2126\*r+0.7152\*g+0.0722\*b; if (newValue>scale) newValue=255; else if (newValue<scale) newValue=0; else if (scale>127.5) newValue=255; else if (scale<127.5) newValue=0;

else newValue=255;

colorArray[j \* width + i] = Color.rgb((int)newValue, (int)newValue, (int)newValue);

returnBitmap.setPixel(i, j, colorArray[j \* width + i]); }

}

### Brightness

Brightness predstavlja filter koji postojeću zadatu sliku osvetljuje ili potamnjuje na osnovu zadatog *scale* parametra koji predstavlja intenzitet. Svaka od vrednosti komponenti *RGB* povećava se za navedeni *scale* faktor i dobija se nova vrednosti komponente. Bitno je napomenuti da se svaka komponenta povećava za isti *scale* faktor pa odnos *RGB* komponenti ostaje isti što dovodi do efekta zatamnjivanja ili osvetljivanja slike dok se same boje ne menjaju. Filter prihvata sliku kao prvi argument, i parametar *scale* koji predstavlja ceo broj od *-255* do *255.* Filter zatim pravi lokalnu kopiju prosleđene slike, a zatim se vrši obilazak matrice i za svaki piksel određuje njegova nova vrednost po formuli.

Pri čemu je *x[i][j]* piksel na poziciji *i,j,* a anotacija *.red, .green, .blue,* označava da se uzima crvena (*R*), zelena (*G*), plava (*B*) komponenta boje tog piksela. Naravno u slučaju prekoračenja opsega *0..255* uzimaju se maksimalne vrednosti *0* i *255*. Dakle ako je *newValue > 255* uzima se da je *x[i][j].red = 255*, ako je *newValue < 0* uzima se da je *x[i][j].red = 0* u suprotnom uzima se *x[i][j].red = newValue.* Analogno se računaju i ostale dve komponente.

if (scale <= 255 && scale >= -255) {  
 for(int i=0; i < width; i++) {  
 for(int j=0; j < height; j++) { r = Color.red(colorArray[j \* width + i]); g = Color.green(colorArray[j \* width + i]); b = Color.blue(colorArray[j \* width + i]);

newValue = r + scale;  
 if (newValue > 255) r = 255; else if (newValue < 0) r = 0; else r = newValue;

newValue = g + scale;  
 if (newValue > 255) g = 255;  
 else if (newValue < 0) g = 0; else g = newValue;

newValue = b + scale;  
 if (newValue > 255) b = 255; else if (newValue < 0) b = 0; else b = newValue;

colorArray[j \* width + i] = Color.rgb(r, g, b); returnBitmap.setPixel(i, j, colorArray[j \* width + i]); }  
 }   
 }

### Contrast

### Flip vertical

Flip vertical predstavlja filter koji postojeću zadatu sliku rotira oko *x* ose za 180 stepeni. Tako dobijena slika je što se intenziteta komponenti *RGB* tiče nepromenjena, ali je okrenuta naopako. Filter prihvata sliku kao prvi argument i jedini argument*.* Filter zatim pravi lokalnu kopiju prosleđene slike, a zatim se vrši obilazak matrice i za svaki piksel određuje njegova nova vrednost tako što se zameni sa odgovarajućim pikselom iz njegove kolone, tj. svaki piksel *x[i][j]*menja se *x[i][height-j-1]* pri čemu je *height* visina slike.

for (int i = 0; i < width; i++) {  
for(int j=0; j < height/2; j++) {  
 int rtemp = Color.red(colorArray[j \* width + i]); int gtemp = Color.green(colorArray[j \* width + i]); int btemp = Color.blue(colorArray[j \* width + i]);

r = Color.red(colorArray[(height-j-1) \* width + i]); g = Color.green(colorArray[(height-j-1) \* width + i]); b = Color.blue(colorArray[(height-j-1) \* width + i]);

colorArray[j \* width + i] = Color.rgb(r, g, b); returnBitmap.setPixel(i, j, colorArray[j \* width + i]);

colorArray[(height-j-1) \* width + i] = Color.rgb(rtemp, gtemp, btemp);  
returnBitmap.setPixel(i, (height-j-1), colorArray[(height-j-1)   
\* width + i]);

}  
}

### Flip horizontal

Flip horizontal predstavlja filter koji postojeću zadatu sliku rotira oko *y* ose za 180 stepeni. Tako dobijena slika je što se intenziteta komponenti *RGB* tiče nepromenjena, ali je dobijen efekat ogledala. Filter prihvata sliku kao prvi argument i jedini argument*.* Filter zatim pravi lokalnu kopiju prosleđene slike, a zatim se vrši obilazak matrice i za svaki piksel određuje njegova nova vrednost tako što se zameni sa odgovarajućim pikselom iz njegove vrste, tj. svaki piksel *x[i][j]*menja se *x[width-i-1][j]* pri čemu je *width* širina slike.

for (int i=0; i < width/2; i++) {  
for (int j = 0; j < height; j++) {  
 int rtemp = Color.red(colorArray[j \* width + i]); int gtemp = Color.green(colorArray[j \* width + i]); int btemp = Color.blue(colorArray[j \* width + i]);

r = Color.red(colorArray[j \* width + (width-i-1)]); g = Color.green(colorArray[j \* width + (width-i-1)]); b = Color.blue(colorArray[j \* width + (width-i-1)]);

colorArray[j \* width + i] = Color.rgb(r, g, b); returnBitmap.setPixel(i, j, colorArray[j \* width + i]);

colorArray[j \* width + (width-i-1)] = Color.rgb(rtemp, gtemp, btemp);  
returnBitmap.setPixel((width-i-1), j,   
colorArray[j \* width + (width-i-1)]);

}  
}

### Grayscale

U fotografiji, pojam *grayscale* ili *grayscale* digitalna slika predstavlja sliku kod koje važi da za svaki piksel vrednosti komponente *RGB* budu međusobno jednake, dakle *R = G = B*. Zbog toga svaki piksel sa sobom nosi jedino informaciju o intenzitetu tih komponenti *RGB* koji mogu biti u opsegu od *0* do *255*. Na taj način se postiže efekat da se *grayscale* slike sastoje samo iz različitih nijansi sive boje, počevši od najtamnije nijanse tj. crne pa sve do najsvetlije bele.

*Grayscale* slike treba razlikovati od *black and white* slika dobjenih primenom filtera *black&white* iz poglavlja 3.1.2 koje se sastoje samo iz dve boje, i to crne i bele bez prelaznih nijansi između njih. Dakle *grayscale* slike imaju različite nijanse sive između najsvetlije bele *(255,255,255)* i najtamnije crne *(0,0,0)*.

Filter za *grayscale* prihvata sliku kao prvi argument i jedini argument*.* Filter zatim pravi lokalnu kopiju prosleđene slike, a zatim se vrši obilazak matrice i za svaki piksel određuje njegova osvetljenost po formuli



Pri čemu je *x[i][j]* piksel na poziciji *i,j,* a anotacija *.red, .green, .blue,* označava da se uzima crvena (*R*), zelena (*G*), plava (*B*) komponenta boje tog piksela. Zatim se ta nova vrednost postavlja za svaku od komponenti *RGB* piksela i na taj način dobija efekat *grayscalea*.

for (int x = 0; x < width; x++) {  
for (int y = 0; y < height; y++) {  
 r = Color.red(colorArray[y \* width + x]); g = Color.green(colorArray[y \* width + x]); b = Color.blue(colorArray[y \* width + x]);

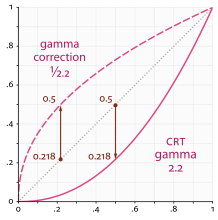
newValue = 0.2126\*r+0.7152\*g+0.0722\*b;   
colorArray[y \* width + x] =   
Color.rgb((int)newValue, (int)newValue, (int)newValue);

returnBitmap.setPixel(x, y, colorArray[y \* width + x]);

}  
}

### Gama korekcija

Pojam gamma potiče još iz doba fotografije i opisuje nelinearnost prikaza slike na fotografskom papiru u odnosu na stvarnu sliku. Pojam korekcije boja kod monitora nasleđen je iz doba nastanka TV sistema. Osobina CRT (cathode ray tube)  sistema je da odnos između upravljačkog signala katodne cevi (signala slike) i intenziteta svetla koji daje piksel nije linearan. Naime, neka 100% vrednosti signala na CRT daje 100% dobijenog intenziteta svetla i neka su to maksimalne normalizovane vrednosti u koordinatnom sistemu. U idealnom sistemu pad signala na 50% vrednosti ima kao rezultat pad intenziteta svetla na 50%. Odnos signala i intenziteta svetla je linearan (slika 3.3). U praksi nije tako, naime pad signala od 50% daje intenzitet svetlosti od oko 25% u odnosu na maksimalnu vrednost. Na slici 3.1. prikazan je primer CRT gama korekcije – linija sa tačkicama predstavlja linearnu funkciju odnosa signala slike (Y osa) i intenziteta svetla (X osa), pri čemu gama faktor iznosi 1. Puna linija prikazuje tipično ponašanje CRT sistema, dok isprekidana linija predstavlja inverznu funkciju, tako da ukupni rezultat daje linearnu zavisnost i sliku prihvatljivu za oko.



3.3 Primer CRT gama korekcije

Gama korekcija je nelinearna operacija koja u svojoj suštini predstavlja posvetljavanje slike u zavisnosti od gama faktora koji predstavlja stepen posvetljavanja slike. Gama korekcija je definisana sledećim izrazom:

*Vout* = *A*\**Vin¥* (3.6)

*A* je konstanta, *Vin* i *Vout* su ulazna i izlazna vrednost koje su nenegativni realni brojevi, a predstavljaju intenzitet vrednosti piksela pre i posle gama korekcije, dok *¥* predstavlja gama faktor koji ukoliko je veći od 1, ima efekat potamnjivanja senki na originalnoj slici, dok ukoliko je u opsegu 0..1 ima efekat posvetljavanja tamnih delova slike.

Prilikom implementacije algoritma za gama korekciju, gama faktor se unosi od strane korisnika, a može biti u opsegu 0..5, pri čemu ukoliko se odabere vrednost 1, originalna slika neće biti izmenjena. Formula na kojoj se bazira implementacija algoritma je sledeća:

*Vout* = *Wmax* \* ((*Vin* – *Wmin*) / (*Wmax* – *Wmin*)) ^ *¥* + *Wmin* (3.7)

gde su *Vin* i *Vout* ulazna i izlazna vrednost piksela, *¥* gama faktor, a *Wmin* i *Wmax* minimalna i maksimalna vrednost intenziteta piksela na slici. Uzeto je da su te vrednosti 0 i 255, respektivno. Najvažniji deo koda koji obavlja gama korekciju nad pikselima prikazan je u tabeli 3.2:

Tabela 3.2: Gama korekcija nad slikom

**final** **int** MAX\_SIZE = 256;

**final** **double** MAX\_VALUE\_DBL = 255.0;

**final** **int** MAX\_VALUE\_INT = 255;

**int**[] gamma = **new** **int**[MAX\_SIZE];

**if** (gammaFactor > 0) {

**for**(**int** i = 0; i < MAX\_SIZE; ++i)

gamma[i] = (**int**)Math.*min*(MAX\_VALUE\_INT,

(**int**)((MAX\_VALUE\_DBL \* Math.*pow*(i / MAX\_VALUE\_DBL, gammaFactor))));

**for**(**int** x = 0; x < width; ++x) {

**for**(**int** y = 0; y < height; ++y) {

pixel = src.getPixel(x, y);

A = Color.*alpha*(pixel);

R = gamma[Color.*red*(pixel)];

G = gamma[Color.*green*(pixel)];

B = gamma[Color.*blue*(pixel)];

bmOut.setPixel(x, y, Color.*argb*(A, R, G, B));

}

}

}

**return** bmOut;

Metoda Color.argb(a, r, g, b) vraća celobrojnu vrednost piksela na osnovu vrednosti pojedinačnih komponenti (alfa komponente, crvene, zelene i plave komponente) koje se prosleđuju kao argumenti poziva metode, a koje treba da budu u opsegu [0..255]. Na slikama 3.3, 3.4, 3.5 i 3.6 predstavljene su originalna slika i slike nakon primene gama korekcije sa gama faktorom od 0.5, 2 i 4.



3.6 Slika sa gama faktorom 4

3.5 Slika sa gama faktorom 2

3.4 Slika sa gama faktorom 0.5

3.3 Originalna slika pre gama korekcije

### Color filter

Color filter nam omogućava filtriranje boja na slici, odnosno prikazivanje slike preko samo jednog kanala boje (crvenog, zelenog ili plavog) ili kombinacijom sva tri kanala boje. Parametri sa kojima se množe pojedinačne komponente piksela (crvena, zelena, plava) unose se od strane korisnika i u opsegu su 0..1. Najvažniji deo koda koji obavlja filtriranje boja na slici prikazan je u tabeli 3.3:

Tabela 3.3: Color filter nad slikom

**for**(**int** x = 0; x < width; ++x) {

**for**(**int** y = 0; y < height; ++y) {

pixel = src.getPixel(x, y);

A = Color.*alpha*(pixel);

R = (**int**)(Color.*red*(pixel) \* red);

G = (**int**)(Color.*green*(pixel) \* green);

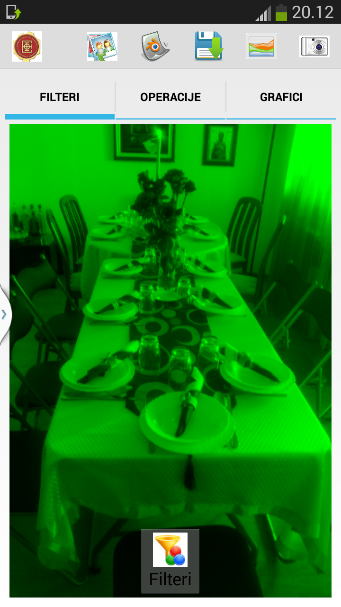
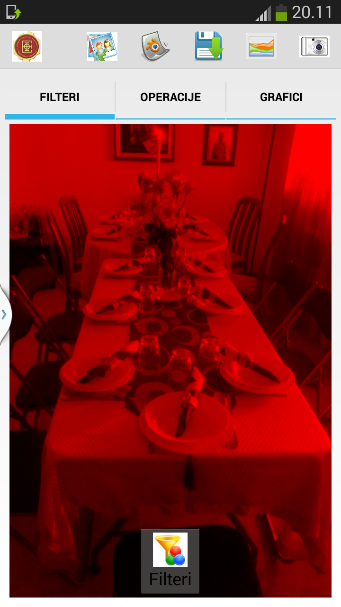
B = (**int**)(Color.*blue*(pixel) \* blue);

bmOut.setPixel(x, y, Color.*argb*(A, R, G, B));

}

}

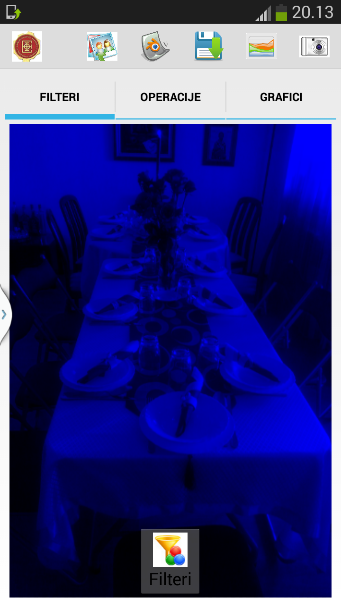
Prikazivanje originalne slike prikazane na slici 3.7 samo preko crvene komponente omogućeno je odabirom RGB parametara koji su (1, 0, 0) kao što je prikazano na slici 3.8; samo preko zelene komponente omogućeno je odabirom RGB parametara koji su (0, 1, 0) kao što je prikazano na slici 3.9; samo preko plave komponente omogućeno je odabirom RGB parametara koji su (0, 0, 1) kao što je prikazano na slici 3.10; preko svih triju komponenti koje su ravnomerno rasporedjene omogućeno je odabirom RGB parametara koji su (0.5, 0.5, 0.5) kao što je prikazano na slici 3.11.



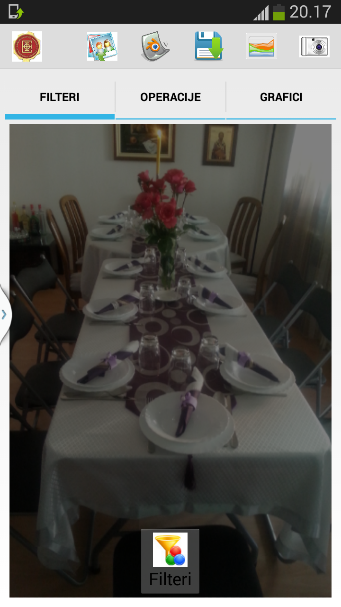
3.8 Slika sa primenom R,G,B = (1, 0, 0)

3.9 Slika sa primenom R,G,B = (0, 1, 0)

3.7 Originalna slika pre color filtera



3.10 Slika sa primenom R,G,B = (0, 0, 1)



3.11 Slika sa primenom R,G,B = (0.5, 0.5, 0.5)

### Shading filter

Shading filter je tehnika koja koristi AND operator izmedju originalne boje piksela i odabrane boje kojom se vrši senčenje slike. Preko dijaloga za odabir boje za senčenje (slika 3.12) korisnik odabira željenu boju kojom će se vršiti senčenje slike. Najvažniji deo koda koji obavlja senčenje slike prikazan je u tabeli 3.4:

Tabela 3.4: Shading filter nad slikom

source.getPixels(pixels, 0, width, 0, 0, width, height);

**int** index = 0;

**for**(**int** y = 0; y < height; ++y) {

**for**(**int** x = 0; x < width; ++x) {

index = y \* width + x;

pixels[index] &= shadingColor;

}

}

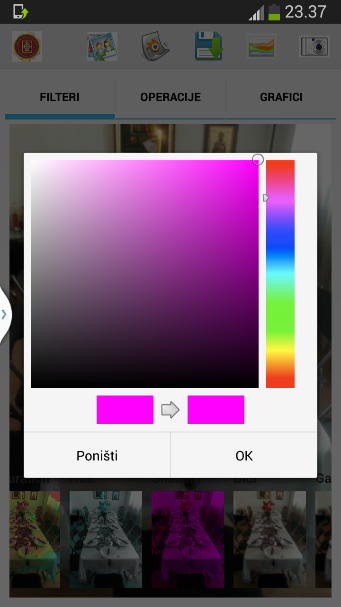
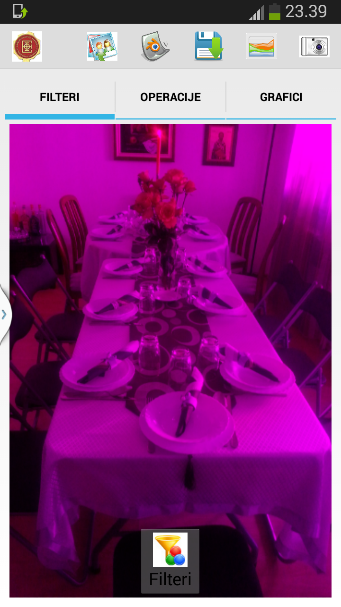
Bitmap bmOut = Bitmap.*createBitmap*(width, height, Bitmap.Config.*ARGB\_8888*);

bmOut.setPixels(pixels, 0, width, 0, 0, width, height);

**return** bmOut;

Pri tome, width predstavlja širinu slike, height dužinu tj. Visinu, a pixels je niz width\*height celih brojeva. Metoda getPixels koja se poziva nad objektom Bitmap-e u nizu pixels vraća kopiju podataka iz bitmap-e. Za svaki piksel računamo poziciju u dvodimenzionalnoj matrici (index) i nad tom vrednošću piksela radimo bitsku operaciju AND sa celobrojnom vrednošću boje kojom senčimo sliku. Format slika ARGB\_8888 označava da se svaki piksel slike skladišti pomoću 4B pri čemu svaka od komponenti piksela zauzima po 8 bitova (R, G i B komponenta i alpha komponenta). Metoda setPixels vrši zamenu svih piksela u bitmap-i bmOut vrednostima piksela uskladištenim u nizu pixels. Na taj način je postignuto da napravimo novu sliku koja će se sastojati od izmenjenih vrednosti piksela originalne slike.

Originalna slika prikazana je na slici 3.13, a senčenje originalne slike izabranom bojom prikazano je na slici 3.14.



3.13 Originalna slika pre shading filtera

3.14 Slika sa shading filterom

3.12 Color Picker dijalog

## Primena binarnih filtera nad slikama

## Primena konvolucionih filtera nad slikom

Konvolucioni filteri su naziv dobili na osnovu matematičke konvolucije, koja predstavlja primenu operacije nad dve funkcije *f* i *g* čime se proizvodi nova fukcina.



Kod procesiranja slika, termin kernel, konvoluciona matrica ili maska predstavljaju matricu koja se primenom konvolucije nad slikom za odgovarajuće vrednosti matrice koje definiše filter dobijaju efekti nad slikama. U samom projektu...

# Primena histograma

# Pregled performansi filtera

# Zaključak

# Literatura