Univerzitet u Sarajevu
Prirodno-matematički fakultet
Teoriska kompjuterska nauka

Seminarski rad

Image Halftoning (polutoniranje slike)

Student: Penjić Safet

Septembar, 2009

Sadržaj

I Uvod

Halftoning (polutoniranje) je korisna tehnika za digitalnu obradu slike i predstavlja proces pretvaranja sive slike u binarnu sliku. Proces polutoniranja se zahtjeva u mnogim današnjim elektronskim aplikacijama kao što su faksimil (FAX), elektronsko skeniranje i kopiranje, lasersko printanje i slično.

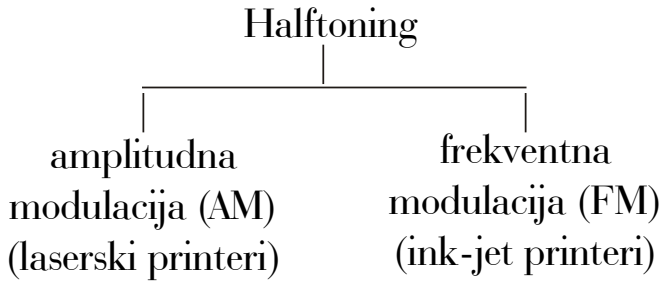
Monitor na sivoj slici piksel sa vrijednošću 255 prikazuje bijelo, piksel sa vrijednošću 0 prikazuje crno, a ostale vrijednosti između 0 i 255 su nijanse od bijele prema crnoj boji, tako da npr. piksel sa vrijednošću 127 predstavlja neku prosječnu sivu boju.

Inkjet printeri (ili skeneri i faks aparati) koji imaju samo crni ketrid mogu odštampati samo crnu boju, a ako ne odštampaju ništa ostaje bijela boja papira. Ovi uređaji nijanse sive boje štampaju tako da crne tačke postavljaju na malom razmaku. Svjetlija područja imaju postavljene crne tačke na većem razmaku od tamnijih područje gdje su ove tačke zbijene.

Halftoning je metoda za kreianje iluzije neprekidne izlazne nijanse na binarnom uređaju.

Imamo dva različita modela za Halftoning:

1. halftoning pomoću amplitudne modulazije (AM)
2. halftoning pomoću frekventne modulacije (FM)



Halftoning pomoću amplitudne modulazije podešavaju veličinu izlazne tačke i koriste ih uglavnom laserski štampači.

Halftoning pomoću frekventne modulacije koriste Inkjet štampači i izlazne tačke kod ovog modela su iste veličine. Tamnija područja se predstavljaju tako da se štampa više ovih tački, dok za svjetlija područje se koristi manja količina (manja frekvencija) tački.

Metode za proizvodnju različitih halftoninga:

1. tačkaste operacije (odnose se na prikaz)
- prikaz pomoću grozda tački (cluster dot screening) (primjer AM)
- prikaz pomoću disperzije tački (disperze dot screening) (primjer FM)

2. lokalno susjedske operacije
- greška širenja (error diffusion) (primjer FM)

3. iteraciska strategija
- direktno binarno traženje (direct binary search – Analoui, Allebach - 1992)

Prednosti tačkastih operacija je taj što su brze, dok je kvalitet slike malo lošiji.

Ovaj seminarski će pokriti tehnike polutoniranja poznate kao *constant treshold* (konstantni prag), *random noise binarization* (binarizacija slučajnim šumom), *ordered dithering* (određeno zamućivanje), *cluster dot screening* (prikaz pomoću grozda tački), *disperze dot screening* (prikaz pomoću raspršivanja tački) i *error diffusion* (greška širenja). Sve problemi će biti urađene u MatLabu.

1.1 Postavka i važnost problema

Pretpostavimo da trebamo isprintati ili skenirati sliku visokog kvaliteta (siva slika ili u boji). Tada postoji mnogo tački koje treba obraditi. Recimo da imamo sliku veličine A4 formata 210×297 mm tj 8,27×11,69" (inča) rezolucije 600 dpi (tački po inču). To znači da imamo sliku veličine 4962×7014 piksela, što je slika koja sadrži 34 803 468 piksela (34 Mpiksela).

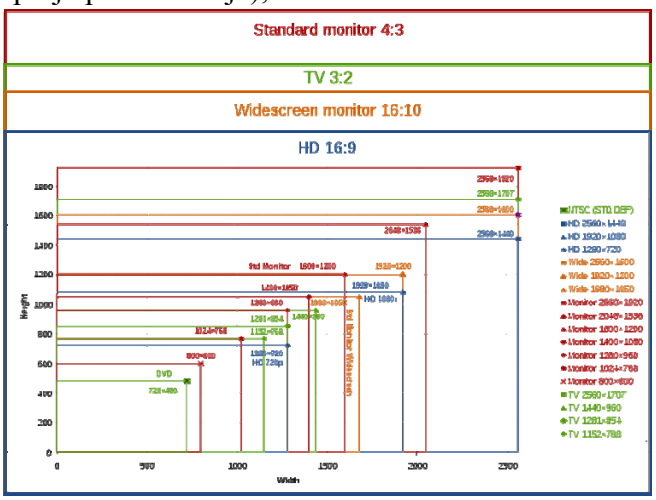
Cilj je isprintati ili skenirati sliku veličine A4 formata u što kračem vremenskom roku, npr jedna stranica po jednoj sekundi.

Recimo da želimo isprintati nešto u boji. Imamo četri boje CMYK i za svaku od ovih boja imamo 8 bita (1 bajt)(256 nijansi) po boji. 34 Mpiksela × 4 B = 136 MB bi imali po jednoj stranici. Koliko vremena treba printeru da isprinta samo jednu stranicu, da ne spominjemo dokument koji ima stotinu stranica. Koliko je krajnji korisnik zadovoljan ako za isprintanu jednu stranicu treba čekati više od jedne minute.

Recimo da želimo kopirati nešto u boji. Postavke su slične kao u prethodnoj priči samo umjesto 4 boje imamo tri (RGB). 34 MB × 3 = 102 MB po jednoj stranici. Koliko vremena treba kopir aparatu da kopira samo jednu stranicu. Koliko je krajnji korisnik zadovoljan ako na kopiju jedne stranice treba čekati više od 15 sekundi. Da li bi se takvi kopir aparati uopšte prodavali?

Printanje, skeniranje i kopiranje nije lako. Poredimo ga sa videom. Trenutni standardi su:

- SDTV: 480i (NTSC, 720×480 podjeljeno u dva polja po 240-linija),
- SDTV: 576i (PAL, 720×576 podjeljeno u dva polja po 288-linija),
- EDTV: 480p (NTSC, 720×480),
- EDTV: 576p (PAL, 720×576),
- HDTV: 720p (1280×720),
- HDTV: 1080i (1920×1080 podjeljeno u dva polja po 540-linija),
- HDTV: 1080p (1920×1080 progressive scan)



Slika 1: Izgled omjera i rezolucija

Standardni EDTV sa NTSC sistemom ima rezoluciju od 720×480 piksela i 30 slika u sekundi.
720×480=345 600 piksela
345600 piksela * 3 boje (RGB) = 1 036 800 piksela
1 036 800 piksela * 30 slika = 31 104 000 bita po sekundi = 31 MB/sek (nekompresovano).

Nekrompesovnom NTSC videu visokog kvaliteta potrebno je 31 MB/sek dok podacima koji izlaze iz kopir aparata treba 102 MB/sek.

Sivoj slici veličine A4 formata rezolucije 600 dpi treba 34 MB, dok binarnoj slici veličine A4 formata treba oko 4 MB.

1.2 Konstantni prag

8-bitna jednobojna slika dozvoljava 256 različitih sivih nivoa. Čak i većina današnjih mobitela dozvoljavaju prikaz ovakvih slika. Međutim neki izlazni uređaji štampaju ili prikazuju slike sa mnogo manje boja. U ovom slučaju sive slike trebaju se pretvoriti u binarne slike, kod kojih su pikseli samo crni ili bijeli.

Najjednostavniji način pretvaranja u binarnu sliku je bazirana na *pragu*, tj dva nivoa (jedan bit) kvantizaciji. Neka je $f(i, j)$ siva slika, i $b(i, j)$ odgovarajuća binarna slika bazirana na pragu. Za dati prag T binarna slika se izračuna pomoću formule:

$$b(i,j) = \begin{cases} 255, \text{ ako je } X(i,j) > T \\ 0, \quad \text{u suprotnom} \end{cases} \tag{1}$$

Figura 1 prikazuju primjer konverzije sive slike u binarnu sliku koristeći konstantni prag, gdje smo za prag uzeli T = 127.

Figura 1:

(a) Orginalna siva slika

(b) Binarna slika napravljena pomoću konstantnog praga

U slikama na Figuri 1 može se vidjeti da binarne slike nisu "osjenčene" ispravno - dobijeni artefekat je poznat pod imenom *netačno ocrtavanje* (*false contouring*). Netačno ocrtavanje se pojavljuje kad je kvantizacija niska, kao u u jedan-kvantizaciji, zato što greška kvantizacije zavisi o ulaznom signalu. Prema tome ako smanjimo ovisnost signala o greški kvantizacije, vizuelna vrijednost binarne slike će biti poboljšana.

1.2.1 Problem broj 1

Učitati RGB sliku i koristiti MatLab za prikaz učitane slike. Pretvoriti učitanu sliku u sivu sliku. Napisati funkciju koja će učitanu sliku pretvoriti u binarnu tehnikom Konstantnog praga, koristeći za prag $T = 127$. Prikazati i isprintati rezultat u zaseban fajl.

Napisati i funkciju koja će izračunati grešku srednje vrijednosti kvadratnog korijena (root mean square error - RMSE), koja je definisana sa:

$$\mathrm{RMSE} = \sqrt{\frac{1}{\mathrm{NM}} \sum_{i,j} \{f(i,j)-b(i,j)\}^2} \qquad (2)$$

gdje je NM ukupan broj piksela na svakoj slici, $f(i, j)$ orginalna slika i $b(i, j)$ binarna slika. Ispisati RMSE poslije kvantizacije date slici.

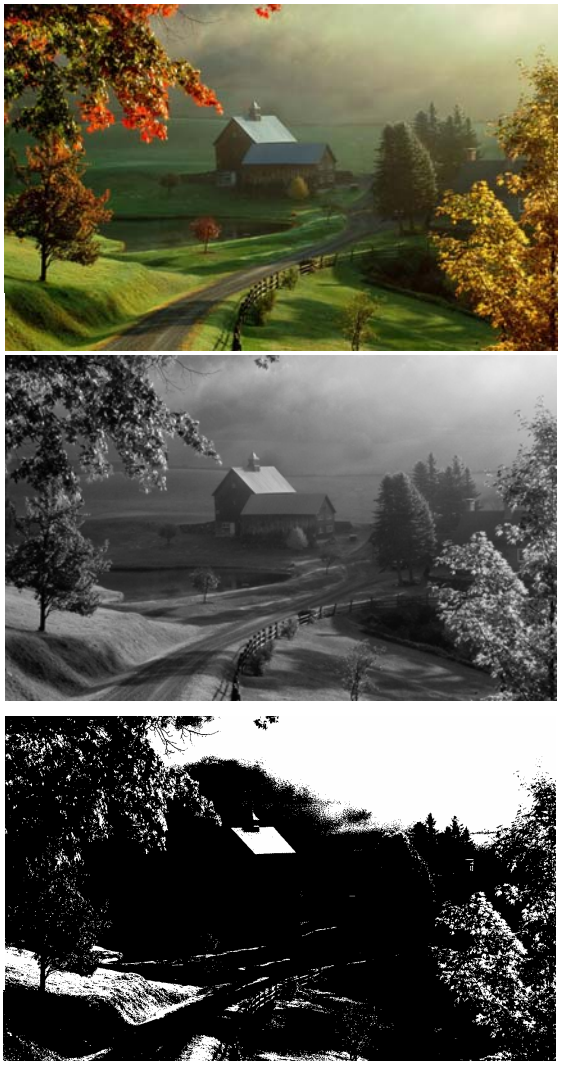
Rješenje:

U fajl tresholding.m ćemo napisati funkciju `tresholding` koja sivu sliku pretvara u binarnu pomoću tehnike konstantnog praga.

tresholding.m:

```
function [ tslika ] = tresholding( cbslika )
%%=======================================
%% TRESHOLDING konvertuje crnobijelu u
binarnu sliku
%% pomocu ktehnike konstantnog praga
%% Objašnjenje: function [ tslika ] =
TRESHOLDING( cbslika )
%% cbslika = ulazna crnobijela slika
%% tslika = izlazna binarna slika
%% ================================
tslika=cbslika;
[visina, sirina]=size(cbslika);
for i=1:1:visina
    for j=1:1:sirina
        if tslika(i,j) > 127
            tslika(i,j)=255;
        else
            tslika(i,j)=0;
        end
    end
end

end
```

```
>> slika=imread('p0018.tif');
>> figure; imshow(slika); axis image;

>> cbslika=rgb2gray(slika);
>> figure; imshow(cbslika); axis image;
>> imwrite(cbslika,'p0018_cb.tif','tif');

>> kpslika=tresholding(cbslika);
>> figure; imshow(kpslika); axis image;
```



Slika 2: (a) orginalna RGB slika (b) siva slika dobijena uz pomoć ugrađene funkcije rgb2gray (c) binarna funkcija dobijena uz pomoć naše funkcije tresholding

```
>> imwrite(kpslika,'p0018_t.tif','tif');
```

U fajl rmse.m ćemo napisati funkciju koja računa grešku srednju vrijednosti kvadratnog korijena.

rmse.m:

```
function [ rezultat ] = rmse(sslika, bslika)
%%=============================================================
%% RMSE računa srednju grešku kvadratnog korijena
%% koja je definisana pomoću formule
%% sqrt((1/MN)*sum{f(i, j)-b(i, j)}^2)
%% Objašnjenje: function [ rezultat ] = RMSE(sslika, bslika)
%% sslika = crnobijela slika
%% bslika = binarna slika
%% rezultat = srednja greška kvadratnog korijena
%% =============================================================
[visina, sirina]=size(sslika);
rezultat = 0;

for i=1:1:visina
    for j=1:1:sirina
        a=double(sslika(i,j));
        b=double(bslika(i,j));
        rezultat = rezultat +(a-b)^2;
    end
end

rezultat = rezultat/(sirina*visina);
rezultat = sqrt(rezultat);
end
```

```
>> rmse_slika = rmse(cbslika, kpslika)
rmse_slika =
  77.9642
```

1.3 Binarizacija slučajnim šumom

Još jedna metoda za smanjivanje ovisnosti signala o greški kvantizacije je dodavanje uniformneog distribuiranog bijelog šuma ulaznoj slici prije kvantizacije. Svakom pikselu na sivoj slici $f(i, j)$, dodaje se slučajan broj n u rangu $[-A, A]$, i tek poslije toga slika koja se dobije kao rezultat se kvantizira jedan-bit kvantizacijom, kao u jednakosti (1) na stranici broj 5. Rezultati ove metode su ilustrirani u Figuri 2, gdje je dodan šum uniformno preko [-128, 128]. Primjetite da iako je rezultirajuća slika nekako kričava, netačno ocrtavanje je dramatično smanjeno.

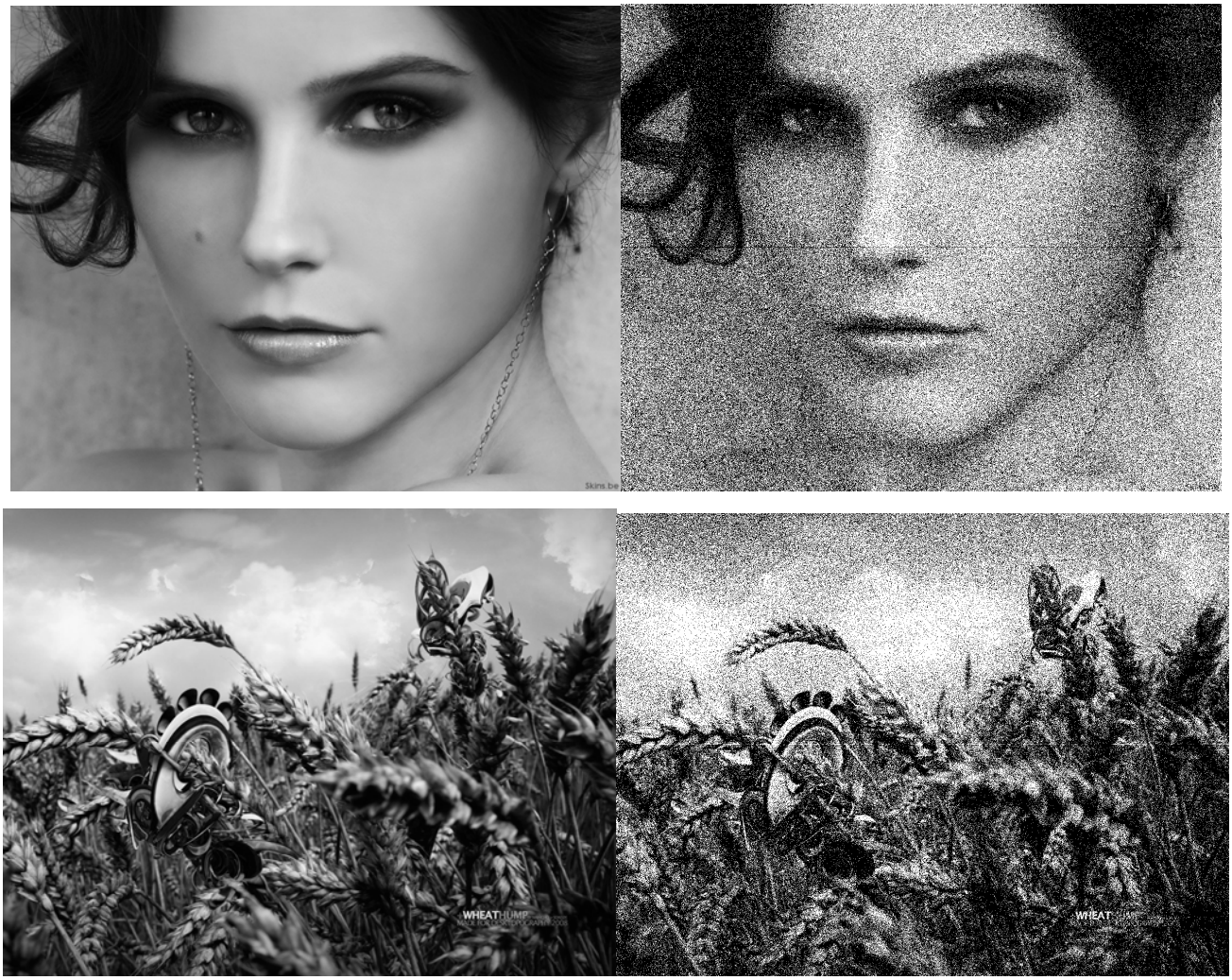


Figura 2:
(a) Orginalna siva slika *(b) Slika dobijena metodom binarizacija slučajnim šumom*

1.3.1 Problem broj 2

Učitati sivu sliku i koristiti MatLab za prikaz učitane slike. Napisati funkciju koja će učitanu sliku pretvoriti u binarnu tehnikom Binarizacija slučajnim šumom. Prikazati i isprintati rezultat u zaseban fajl. Ispisati RMSE poslije kvantizacije date slici.

Rješenje:

U fajlu rnoise.m se nalazi funkcija koja crnobijelu sliku pretvara u binarnu tehnikom binarizacija slučajnim šumom.

rnoise.m

```
function [ rnslika ] = rnoise( cbslika )
%% ============================================
%% RNOISE konvertuje crnobijelu u binarnu sliku
%% pomoću metode binarizacija slučajnog šuma
%% Objašnjenje: function [ RNOISE ] = rnoise( slika )
%% cbslika = ulazna crnobijela slika sa 256 boja
%% rnslika = izlazna binarna slika
%% ================================================
rnslika=cbslika;
[visina, sirina]=size(cbslika);
for i=1:1:visina
    for j=1:1:sirina
```

```
        r = ceil(random('unif',-
128,128));
        rnslika(i,j)=rnslika(i,j)+r;
        if rnslika(i,j) > 127
            rnslika(i,j)=255;
        else
            rnslika(i,j)=0;
        end
    end
end

end
```

```
>> slika=imread('p0018_cb.tif');
>> figure; imshow(slika);  axis image;

>> rnslika=rnoise(slika);
>> figure; imshow(rnslika);  axis image;
>> imwrite(rnslika,'p0018_rn.tif','tif');

>> rmse_slika=rmse(slika,rnslika)

rmse_slika =
  110.2221
```

Slika 3
(a) orginalna siva slika
(b) slika dobijena uz pomoć naše funkcije rnoise

1.4 Zadatak za vježbu

Učitati neku sivu sliku. Napisati funkciju koja će učitanu sliku pretvoriti u binarnu tehnikom Konstantnog praga i tehnikom Binarizacija slučajnim šumom koja će za prag koristiti srednji nivo sivog za datu sliku. Prikazati i isprintati rezultat u zaseban fajl. Ispisati RMSE poslije kvantizacije date slici.

II Određeno zamućivanje

Halftone slike su binarne slike koje bi svojim prikazom trebale izgledati kao sive slike. Iako tehnika slučajnog praga opisana u dijelu 3 glave I može biti korištena za pravljenje halftone slika, nije često korištena u realnim aplikacijama zbog proizvodnje vrlo šumovitih rezultata. U ovoj glavi opisat čemo drugu halftoning tehniku poznatu pod imenom određeno zamućivanje (*ordered dithering*).

Ljudski vizelni sistem teži prosječnom regionu oko piksela umjesto tretiranja svakog piksela zasebno, pa je moguće kreirati iluziju sivih nivoa u binarnoj slici, iako u stvarnosti postoje samo dva siva nivoa. Sa 2×2 binarnom mrežom piksela, možemo predstaviti 5 različitih "efektivnih" svjetlosnih nivoa, kao što je ilustrirano na figuri 3. Slično sa 2×2 binarnom mrežom možemo predstaviti 10 različitih sivih nivoa. Kada želimo zamutiti sliku, mjenjamo blokove orginalne slike sa ovakvim tipovima binarnih mrežnih uzoraka:

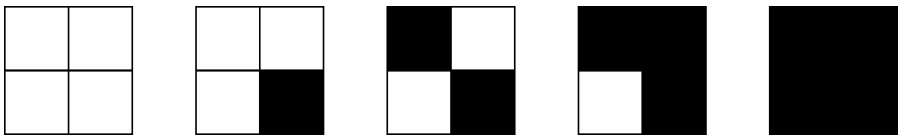


Figura 3: Pet različitih uzoraka od 2×2 binarne mreže piksela

Sjetimo se iz dijela dijela 2 glave I da artefekt netačnog ocrtavanja može biti smanjeno ovisnost signala o greški kvantizacije. Pokazali smo da dodavanjem uniformnog šuma jednobojnoj slici može biti korišteno da postigne ovu dekorelaciju. Alternativni metod bi bio da koristimo promjenjivi prag za proces kvantizacije.

Određeno zamučivanje se sastoji od upoređivanja blokova orginalne slike sa 2D mrežom, poznatom pod imenom *uzorak zamučivanja*. Svaki element bloka se kvantizira koristeći odgovarajuću vrijednost u uzorku zamučivanja kao prag. Vrijednosti u matrici zamučivanja su fiksirani, ali su tipično različiti jedni od drugih. Iz razloga što se prag mijenja između susjednih piksela, neke dekorelacije iz greške kvantizacije su postignute, što ima efekat smanjivanja artefekta netačnog ocrtavanja.

Matricu uzorka možemo definisat i nazvati *indeks matrica*. Indeks matrica određuje poredak u kojima će se tačke uključivati kad slika postaje tamnija (za veću apsorpciju). Na primjer u Figuri 3, indeks matrica je data sa

$$I_2(i, j) = \begin{bmatrix} 1 & 2 \\ 3 & 0 \end{bmatrix} \tag{3}$$

gdje 0 predstavlja prvi piksel koji se treba uključiti a 3 predstavlja zadnji piksel koji se treba uključiti.

Za svaku indeks matricu, postoji odgovarajuća prag matrica koja se koristi za polutoniranje slike. Prag matrica može biti određena iz indeks matrice $I(i, j)$ preko relacije

$$T(i, j) = 255\ \frac{I(i, j) + 0{,}5}{N^2} \tag{4}$$

gdje je N^2 ukupan broj elemenata u indeks matrici. Prag matrica se koristi za polutoniranje slike upoređivanjem svakog piksela sa tačkom u prag matrici. S obzirom da je slika mnogo veća od praag matrice, matrica se ponavlja periodično ili se poploča preko čitave slike. Konkretno, ovo se uradi pomoču operacije

$$b(i,j) = \begin{cases} 255, \text{ ako je } X(i,j) > T((i \bmod N)+1, (j \bmod N)+1) \\ 0, \quad \text{u suprotnom} \end{cases} \quad (5)$$

2.1 Prikaz pomoću grozda tački

Definicija: Ako su uzastopni pragovi locirani u prostornoj blizini, tada ovo zovemo "klaster tačkasti prikaz". Indeks matrica za 4 × 4:

14	11	4	15
10	3	0	5
9	2	1	6
13	8	7	12

Indeks matrica za 8 × 8 mrežu:

62	57	48	36	37	49	58	63
56	47	35	21	22	38	50	59
46	34	20	10	11	23	39	51
33	19	9	3	0	4	12	24
32	18	8	2	1	5	13	25
45	31	17	7	6	14	26	40
55	44	30	16	15	27	41	52
61	54	43	29	28	42	53	60

			3	0			
			2	1			

				10	11		
			9	3	0	4	12
			8	2	1	5	13
				7	6	14	
				16	15		

				21	22		
			20	10	11	23	
	19	9	3	0	4	12	24
32	18	8	2	1	5	13	25
	31	17	7	6	14	26	
		30	16	15	27		
			29	28			

		48	36	37	49		
	47	35	21	22	38	50	
46	34	20	10	11	23	39	51
33	19	9	3	0	4	12	24
32	18	8	2	1	5	13	25
45	31	17	7	6	14	26	40
	44	30	16	15	27	41	
		43	29	28	42		

Primjetite da se tačke uključuju u kružnom toku. Indeks matrica za 16 × 16 mrežu:

254	249	240	227	210						190	211	228	241	250	255
248	239	226	209								191	212	229	242	251
238	225	208										192	213	230	243
224	207												193	214	231
206				62	57	48	36	37	49	58	63			194	215
				56	47	35	21	22	38	50	59				195
				46	34	20	10	11	23	39	51				
				33	19	9	3	0	4	12	24				
				32	18	8	2	1	5	13	25				
				45	31	17	7	6	14	26	40				
				55	44	30	16	15	27	41	52				
205				61	54	43	29	28	42	53	60				196
223	204													197	216
237	222	203											198	217	232
247	236	221	202									199	218	233	244
253	246	235	220	201							200	219	234	245	252

254	249	240	227	210	189	164	135	136	165	190	211	228	241	250	255
248	239	226	209	188	163	134	104	105	137	166	191	212	229	242	251
238	225	208	187	162	133	103	77	78	106	138	167	192	213	230	243
224	207	186	161	132	102	76	54	55	79	107	139	168	193	214	231
206	185	160	131	101	75	53	36	37	56	80	108	140	169	194	215
184	159	130	100	74	52	35	21	22	38	57	81	109	141	170	195
158	129	99	73	51	34	20	10	11	23	39	58	82	110	142	171
128	98	72	50	33	19	9	3	0	4	12	24	59	83	111	143
127	97	71	49	32	18	8	2	1	5	13	25	40	60	84	112
157	126	96	70	48	31	17	7	6	14	26	41	61	85	113	144
183	156	125	95	69	47	30	16	15	27	42	62	86	114	145	172
205	182	155	124	94	68	46	29	28	43	63	87	115	146	173	196
223	204	181	154	123	93	67	45	44	64	88	116	147	174	197	216
237	222	203	180	153	122	92	66	65	89	117	148	175	198	217	232
247	236	221	202	179	152	121	91	90	118	149	176	199	218	233	244
253	246	235	220	201	178	151	120	119	150	177	200	219	234	245	252

Recimo da želimo proizvesti 256 nivoa sive boje. Tad nam treba 16×16 niz piksela (16*16=256). Za vrijednost nula nečemo printati ništa. Za jedan printamo jednu tačku. Za dva printamo dvije tačke, i tako dalje. Možemo proizvesti 256 sivih nivoa iz ovog niza. Ako imamo štampač koji treba neku sliku odštampati, recimo da je ta slika rezolucije 1024×1024, kako ne možemo predstaviti svih 1024 piksela ni u širini ni u visini, treba nam neki uzotrak. Recimo da je njegova veličina npr 16×16. Tako će slika rezolucije 1024×1024 poslije tehnike polutoniranja postati slika rezolucije 64×64. Ako uzorak uzmemo manji, npr 8×8, tad imamo 64 nivoa sive boje i sliku rezolucije 128×128. Tako da možemo razmjeniti broj sivih nivoa sa rezolucijom. Ali to nije baš poželjna razmjena. Kao korisnici digitalne obrade slike želimo da imamo i visoku reoluciju i veliki broj sivih nivoa. U printanju sa uzorcima možemo imati malu rezoluciju i veliki broj sivih nivoa ili veliku rezoluciju i mali broj sivih nivoa.

Prikaz pomoču grozda tački ima relativno vidljive teksture, relativno slabe detalje interpretacije slike i uniformne teksture preko čitave sive skale.

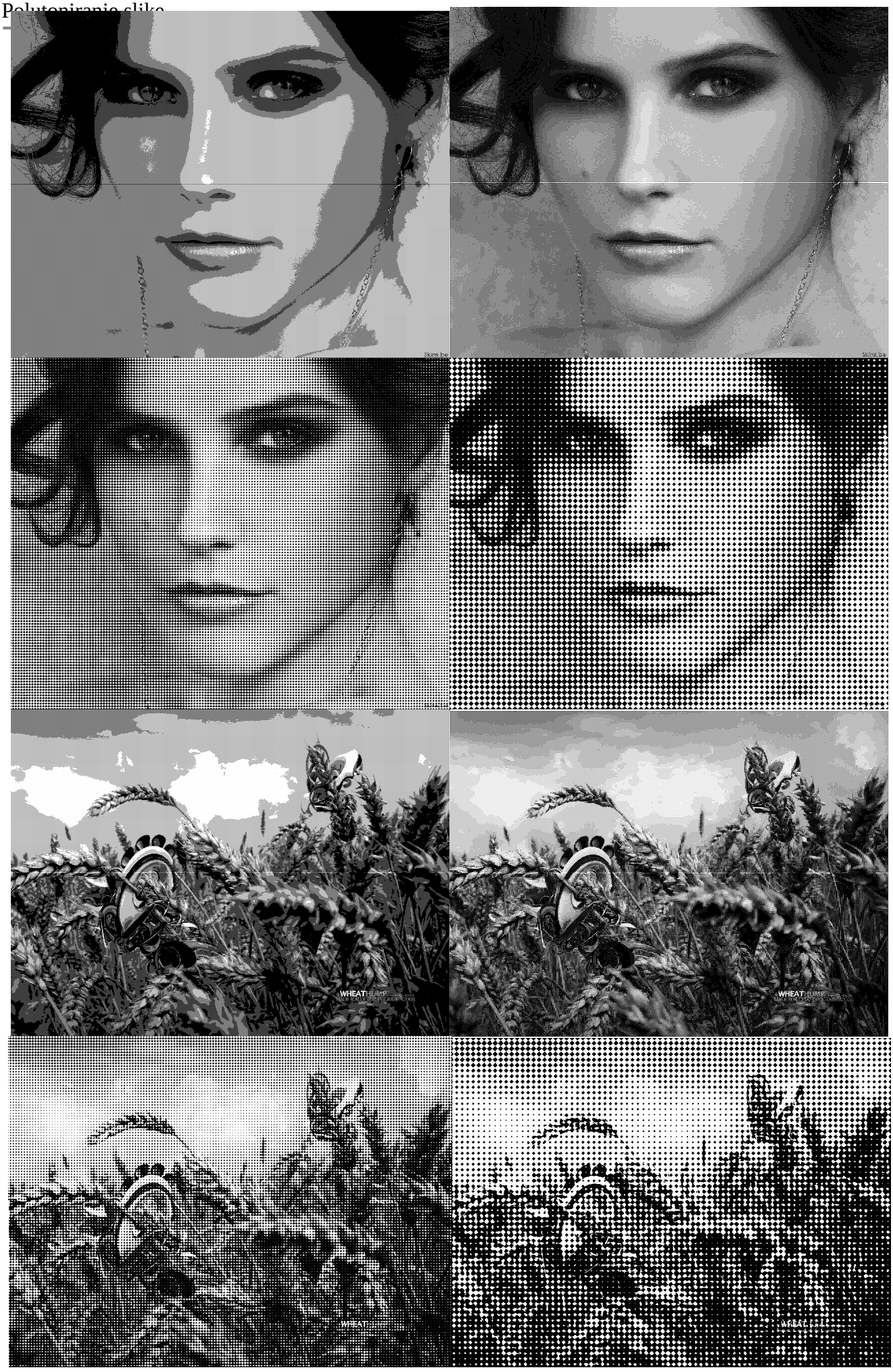
Ima dobru izvedbu na ne-idealnom izlaznom uređaju

- ne postoji preklapanje tački
- mrlja u mrlju promjenjivost
- šum

Figura 3 prikazuje polutonirane slike proizvedene tehnikom grozda tački čije su indeks matrica veličine 2, 4, 8 i 16 (prikazane ranije).

Figura 3: Slike dobijene tehnikom grozda tački čije su indeks matrice redom veličine 2, 4, 8 i 16

Polutoniranje slike



2.1.1 Problem broj 3

Napraviti Cluster dot prag matrice veličina 2×2, 4×4, 8×8 i 16×16. Napisati funkciju koja će generisati četri različita polutona primjenjujući svaku od prag matrica na datu sliku. Isprintati sve četri rezultata u zasebne fajlove.

Za svaku od četri polutona izračunati srednju vrijednost greške kvadratnog korijena između polutone i orginalne slike.

Ljudski vizuelni sistem približno se ponaša kao nisko-propusni filter. Ovaj smireni efekat pravi polutoniranu i orginalnu sliku vrlo sličnom. U zahtjevu da napravimo što precizniju mjeru sličnosti, izračunajte srednju vrijednost greške kvadratnog korijena između orginalne slike, i filtrirane verzije binarne polutonirane slike. Ova nova mjera sličnosti ćemo označiti kao greška srednjeg težinskog kvadratnog korijena (*root weighted mean squared error* - RWMSE). Koristiti 7×7 Gaussian filter definisan pomoću sljedeće funkcije tačke širenja:

$$h(i,j) = \begin{cases} C\exp\left(-\frac{i^2+j^2}{2\lambda}\right), & \text{za } |i| \le 3 \text{ i } |j| \le 3 \\ 0, & \text{u suprotnom} \end{cases} \qquad (6)$$

gdje je λ = 2 i C je normalizirana konstanta takva da je $\sum_{i,j} h(i,j) = 1$.

Rješenje:
Cluster dot prag matrice veličina 2×2, 4×4, 8×8 i 16×16 su:

2×2:

223.1250 31.8750
95.6250 159.3750

4×4:

231.0938 183.2813 71.7188 247.0313
167.3438 55.7813 7.9688 87.6563
151.4063 39.8438 23.9063 103.5938
215.1563 135.4688 119.5313 199.2188

8×8:

249.0234 229.1016 193.2422 145.4297 149.4141 197.2266 233.0859 253.0078
225.1172 189.2578 141.4453 85.6641 89.6484 153.3984 201.2109 237.0703
185.2734 137.4609 81.6797 41.8359 45.8203 93.6328 157.3828 205.1953
133.4766 77.6953 37.8516 13.9453 1.9922 17.9297 49.8047 97.6172
129.4922 73.7109 33.8672 9.9609 5.9766 21.9141 53.7891 101.6016
181.2891 125.5078 69.7266 29.8828 25.8984 57.7734 105.5859 161.3672
221.1328 177.3047 121.5234 65.7422 61.7578 109.5703 165.3516 209.1797
245.0391 217.1484 173.3203 117.5391 113.5547 169.3359 213.1641 241.0547

16×16:

kolone od 1 do 8
253.5059 248.5254 239.5605 226.6113 209.6777 188.7598 163.8574 134.9707
247.5293 238.5645 225.6152 208.6816 187.7637 162.8613 133.9746 104.0918
237.5684 224.6191 207.6855 186.7676 161.8652 132.9785 103.0957 77.1973
223.6230 206.6895 185.7715 160.8691 131.9824 102.0996 76.2012 54.2871
205.6934 184.7754 168.8379 130.9863 101.1035 75.2051 53.2910 36.3574
183.7793 158.8770 129.9902 100.1074 74.2090 52.2949 35.3613 21.4160
157.8809 128.9941 99.1113 73.2129 51.2988 34.3652 20.4199 10.4590
127.9980 98.1152 72.2168 50.3027 33.3691 19.4238 9.4629 3.4863

```
  127.0020   97.1191   71.2207   49.3066   32.3730   18.4277    8.4668    2.4902
  156.8848  126.0059   96.1230   70.2246   48.3105   31.3770   17.4316    7.4707
  182.7832  155.8887  125.0098   95.1270   69.2285   47.3145   30.3809   16.4355
  204.6973  181.7871  154.8926  124.0137   94.1309   68.2324   46.3184   29.3848
  222.6270  203.7012  180.7910  153.8965  123.0176   93.1348   67.2363   45.3223
  236.5723  221.6309  202.7051  179.7949  152.9004  122.0215   92.1387   66.2402
  246.5332  235.5762  220.6348  201.7090  178.7988  151.9043  121.0254   91.1426
  252.5098  245.5371  234.5801  219.6387  200.7129  177.8027  150.9082  120.0293
 kolone od 9 do 16
  135.9668  164.8535  189.7559  210.6738  227.6074  240.5566  249.5215  254.5020
  105.0879  136.9629  165.8496  190.7520  211.6699  228.6035  241.5527  250.5176
   78.1934  106.0840  137.9590  166.8457  191.7480  212.6660  229.5996  242.5488
   55.2832   79.1895  107.0801  138.9551  167.8418  192.7441  213.6621  230.5957
   37.3535   56.2793   89.1504  108.0762  139.9512  168.8379  193.7402  214.6582
   22.4121   38.3496   57.2754   81.1816  109.0723  140.9473  169.8340  194.7363
   11.4551   23.4082   39.3457   58.2715   82.1777  110.0684  141.9434  170.8301
    0.4980    4.4824   12.4512   24.4043   59.2676   83.1738  111.0645  142.9395
    1.4941    5.4785   13.4473   25.4004   40.3418   60.2637   84.1699  112.0605
    6.4746   14.4434   26.3965   41.3379   61.2598   85.1660  113.0566  143.9355
   15.4395   27.3926   42.3340   62.2559   86.1621  114.0527  144.9316  171.8262
   28.3887   43.3301   63.2520   87.1582  115.0488  145.9277  172.8223  195.7324
   44.3262   64.2480   88.1543  116.0449  146.9238  173.8184  196.7285  215.6543
   65.2441   89.1504  117.0410  147.9199  174.8145  197.7246  216.6504  231.5918
   90.1465  118.0371  148.9160  175.8105  198.7207  217.6465  232.5879  243.5449
  119.0332  149.9121  176.8066  199.7168  218.6426  233.5840  244.5410  251.5137
```

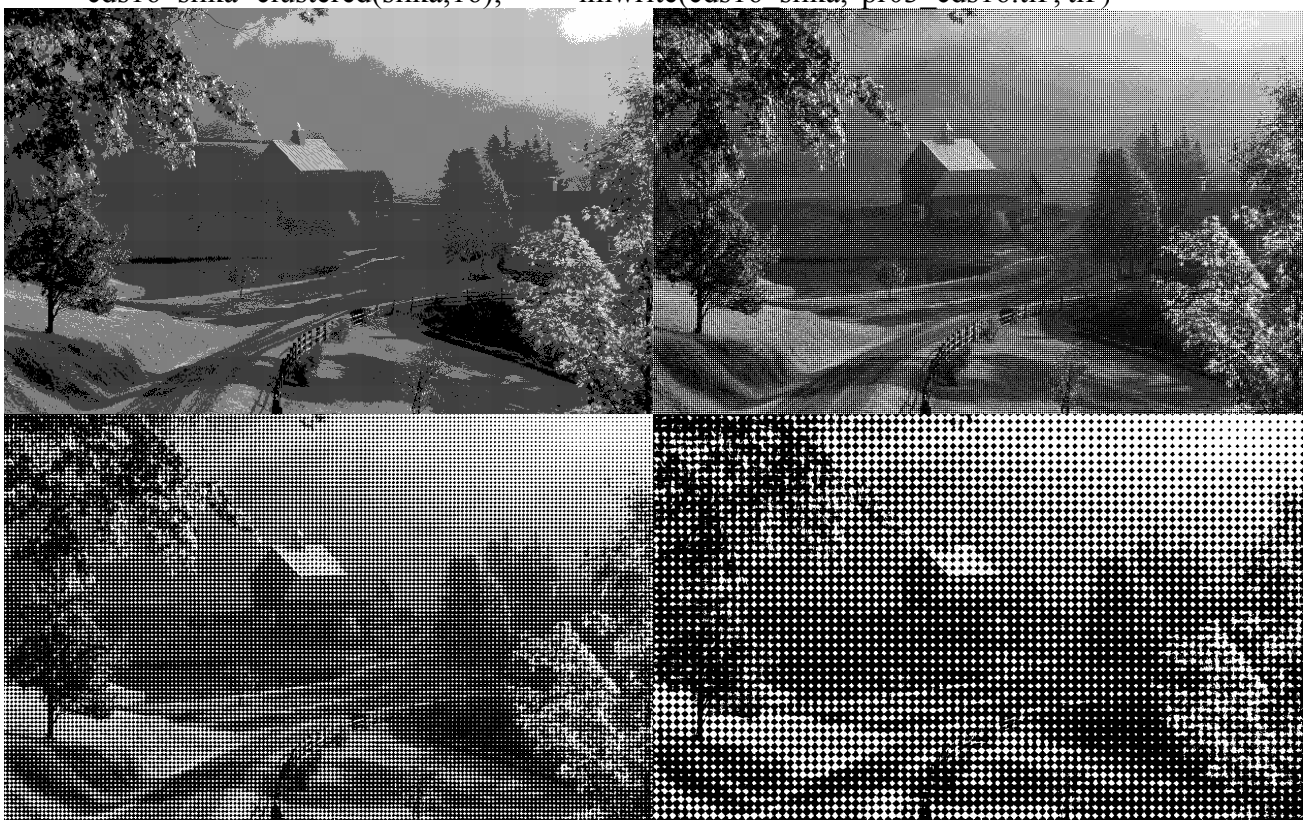
U fajlu clustered.m nalazi se funkcija koja vraća binarnu sliku dobijenu uz pomoć tehnike grozda tački:

```
function [ cds_slika ] = clustered( sslika, broj )
%% ================================================
%% CLUSTERED pretvara sivu sliku u binarnu
%% pomoću tehnike prikaz pomoću grozda tački
%% sslika = ulazna siva slika
%% broj = neki od brojeva 2, 4, 8, ili 16
%% cds_slika = izlazna slika
%% ================================================

[visina, sirina]=size(sslika);
if broj == 2
    prag_matrica = [3 0; 1 2];
else
    if broj == 4
       prag_matrica = [14 11 4 15; 10 3 0 5; 9 2 1 6; 13 8 7 12];
    else
       if broj == 8
           prag_matrica = [ 62 57 48 36 37 49 58 63;
                            56 47 35 21 22 38 50 59;
                            46 34 20 10 11 23 39 51;
                            33 19  9  3  0  4 12 24;
                            32 18  8  2  1  5 13 25;
                            45 31 17  7  6 14 26 40;
                            55 44 30 16 15 27 41 52;
                            61 54 43 29 28 42 53 60 ];
       else if broj == 16
prag_matrica = [254 249 240 227 210 189 164 135 136 165 190 211 228 241 250 255;
248 239 226 209 188 163 134 104 105 137 166 191 212 229 242 251;
238 225 208 187 162 133 103  77  78 106 138 167 192 213 230 243;
224 207 186 161 132 102  76  54  55  79 107 139 168 193 214 231;
```

```
206 185 169 131 101  75  53  36  37  56  89 108 140 169 194 215;
184 159 130 100  74  52  35  21  22  38  57  81 109 141 170 195;
158 129  99  73  51  34  20  10  11  23  39  58  82 110 142 171;
128  98  72  50  33  19   9   3   0   4  12  24  59  83 111 143;
127  97  71  49  32  18   8   2   1   5  13  25  40  60  84 112;
157 126  96  70  48  31  17   7   6  14  26  41  61  85 113 144;
183 156 125  95  69  47  30  16  15  27  42  62  86 114 145 172;
205 182 155 124  94  68  46  29  28  43  63  87 115 146 173 196;
223 204 181 154 123  93  67  45  44  64  88 116 147 174 197 216;
237 222 203 180 153 122  92  66  65  89 117 148 175 198 217 232;
247 236 221 202 179 152 121  91  90 118 149 176 199 218 233 244;
253 246 235 220 201 178 151 120 119 150 177 200 219 234 245 252 ];
            end
        end
    end
end
prag_matrica = 255*((prag_matrica+0.5)/(broj^2));
cds_slika = sslika;
for i=1:1:visina
    for j=1:1:sirina
        m = (mod(i,broj))+1;
        n = (mod(j,broj))+1;
        if sslika(i,j) > prag_matrica(m,n);
           cds_slika(i,j)=255;
        else
           cds_slika(i,j)=0;
        end
    end
end
end
```

Kod koji kucamo u MatLabu:

```
>> slika=imread('pr03_cb.tif');
>> cds2_slika=clustered(slika,2);      >> imwrite(cds2_slika, 'pr03_cds2.tif','tif')
>> cds4_slika=clustered(slika,4);      >> imwrite(cds4_slika, 'pr03_cds4.tif','tif')
>> cds8_slika=clustered(slika,8);      >> imwrite(cds8_slika, 'pr03_cds8.tif','tif')
>> cds16_slika=clustered(slika,16);    >> imwrite(cds16_slika, 'pr03_cds16.tif','tif')
```



Slika 4: Polutonirane slike dobijene uz pomoć Cluster dot prag matrica veličina (a) 2×2 (b) 4×4 (c) 8×8 (d) 16×16

Dobijene polutonirane slike se nalaze iznad. Izračunajmo grešku srednje vrijednosti kvadratnog korijena:

```
>> rmse_s2=rmse(slika, cds2_slika)
rmse_s2 =
  110.9382
>> rmse_s4=rmse(slika, cds4_slika)
rmse_s4 =
  110.1491
>> rmse_s8=rmse(slika, cds8_slika)
rmse_s8 =
  110.1077
>> rmse_s16=rmse(slika, cds16_slika)
rmse_s16 =
  109.7772
```

Napišimo program koji će naći konstantu C i matricu tipa 7×7 koja predstavlja Gausov filter definisan pomoću sljedeće funkcije tačke širenja

$$h(i,j)=\begin{cases} C\exp(-\dfrac{i^2+j^2}{2\lambda}), & \text{za } |i|\le 3 \text{ i } |j|\le 3 \\ 0, & \text{u suprotnom} \end{cases} \qquad (7)$$

U fajlu gausovFilter.m se nalazi program koji kao izlaz ima matricu koja predstavlja gausov filter:

```
gaus=ones(7,7);
for i=1:1:7
    for j=1:1:7
        gaus(i, j)=(exp(-((i-4)^2+(j-4)^2)/4));
    end
end
c=0.09;
trazena=gaus*c;
while(true)
    c=c-0.000001;
    trazena=gaus*c;

if(sum(trazena(1,:))+sum(trazena(2,:))+sum(trazena(3,:))+sum(trazena(4,:))+sum(tr
azena(5,:))+sum(trazena(6,:))+sum(trazena(7,:))<1.0000)

        gaus=trazena;

suma=sum(trazena(1,:))+sum(trazena(2,:))+sum(trazena(3,:))+sum(trazena(4,:))+sum(
trazena(5,:))+sum(trazena(6,:))+sum(trazena(7,:))
        c
        gaus
        break;
    end
end
```

Kad ukucamo:

```
>> gausovFilter
```

Kao rezultat imamo:

```
suma =
   1.0000
c =
   0.0814

gaus =
   0.0009   0.0032   0.0067   0.0086   0.0067   0.0032   0.0009
   0.0032   0.0110   0.0233   0.0300   0.0233   0.0110   0.0032
   0.0067   0.0233   0.0494   0.0634   0.0494   0.0233   0.0067
   0.0086   0.0300   0.0634   0.0814   0.0634   0.0300   0.0086
   0.0067   0.0233   0.0494   0.0634   0.0494   0.0233   0.0067
   0.0032   0.0110   0.0233   0.0300   0.0233   0.0110   0.0032
   0.0009   0.0032   0.0067   0.0086   0.0067   0.0032   0.0009
```

Primjenimo ovaj filter na sve dobijene polutonirane slike:

Funkcija primjeniFilter.m:

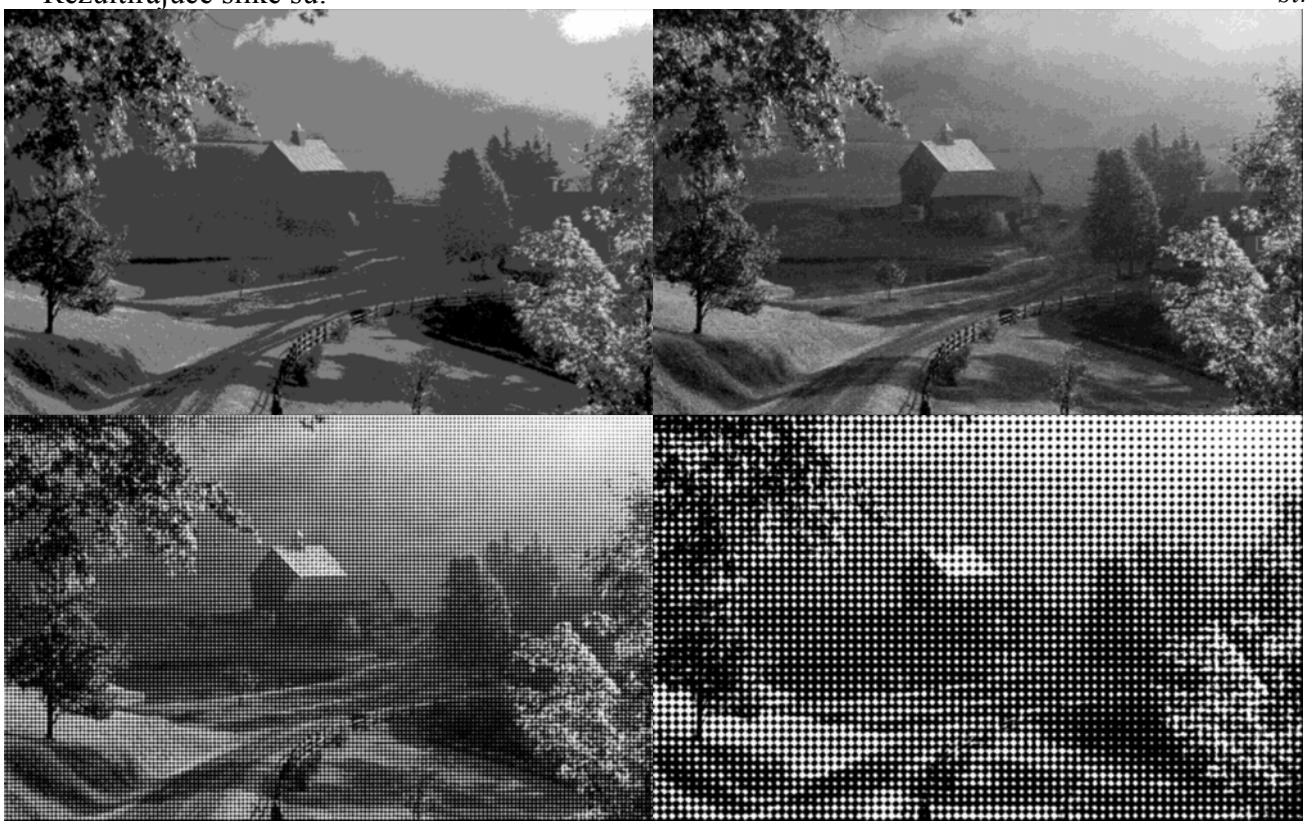
```
function [ filtriranaSlika ] = primjeniFilter (slika, filter)
%% ============================================================
%% PRIMJENIFILTER primjenjuje filter na datu sivu sliku
%% Objašnjenje: function [filtriranaSlika] = primjeniFilter (slika, filter)
%% slika = ulazna siva slika
%% filter = filter u obliku matrice koji želimo primjeniti
%% filtriranaSlika = rezultirajuća slika
%% ============================================================
[m, n]=size(slika);
F = fft2(slika, 2*m, 2*n);
H = fft2(filter, 2*m, 2*n);
G = H.*F;
filtriranaSlika=real(ifft2(G));
filtriranaSlika=filtriranaSlika(5:m+4, 5:n+4);
filtriranaSlika=uint8(filtriranaSlika);
end
```

Slika 5: Slike dobijene kad se na polutonirane slike čije su prag matricce bile veličina 2×2, 4×4, 8×8 i 16×16 primjeni Gaussov filter definisan sa (7) na prethodnoj strani

Kucamo redom:

```
>> f2_slika=primjeniFilter(cds2_slika, gaus);    >> imwrite(f2_slika,'f_pr03_2.tif','tif');
>> f4_slika=primjeniFilter(cds4_slika, gaus);    >> imwrite(f4_slika,'f_pr03_4.tif','tif');
>> f8_slika=primjeniFilter(cds8_slika, gaus);    >> imwrite(f8_slika,'f_pr03_8.tif','tif');
>> f16_slika=primjeniFilter(cds16_slika, gaus); >> imwrite(f16_slika,'f_pr03_16.tif','tif');
```

Rezultirajuće slike su:



Izračunajmo još grešku srednjeg težinskog kvadratnog korijena (*root weighted mean squared error* - RWMSE):

```
>> rwmse_s2=rmse(slika, f2_slika)
rwmse_s2 =   20.8829

>> rwmse_s4=rmse(slika, f4_slika)
rwmse_s4 =   19.3083

>> rwmse_s8=rmse(slika, f8_slika)
rwmse_s8 =   51.5289

>> rwmse_s16=rmse(slika, f16_slika)
rwmse_s16 =   80.8808
```

2.2 Prikaz pomoću raspršivanja tački

Indeks matrica prikazana sa (2) na 10 stranici je specijalni slučaj familije matrica zamučivanja koje je prvi definisao Bayer [1]. Bayer index matrice su definisane rekurzivno pomoću formulu:

$$I_{2n}(i,j) = \begin{bmatrix} 4 * I_n + 1 & 4 * I_n + 2 \\ 4 * I_n + 3 & 4 * I_n \end{bmatrix} \qquad (8)$$

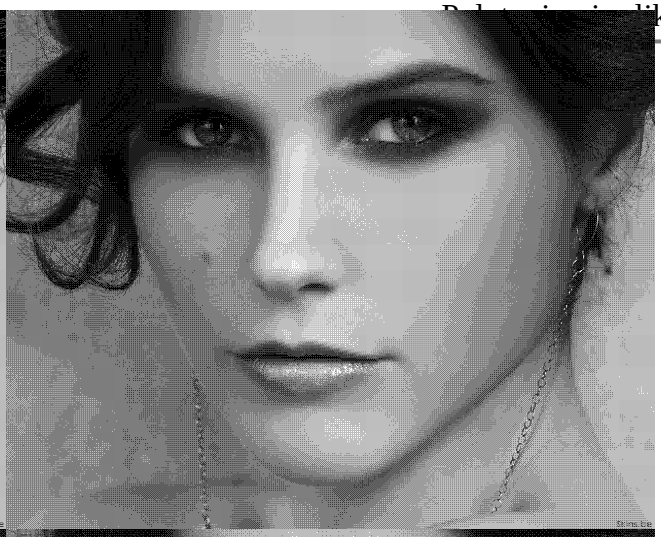
gdje je I_{2n} nova $(2N) \times (2N)$ matrica a I_n je stara $N \times N$ matrica. Ova operacija može biti izvedena sa sljedećom MatLab naredbom:

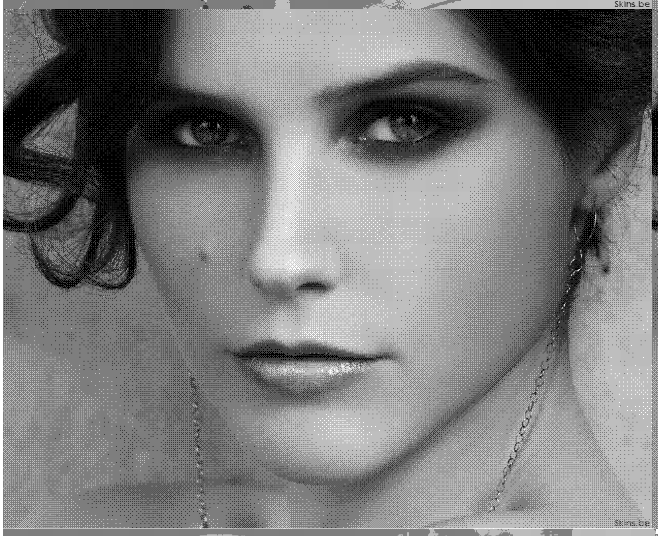
```
I2N = [4*IN+1, 4*IN+2; 4*IN+3, 4*IN];
```

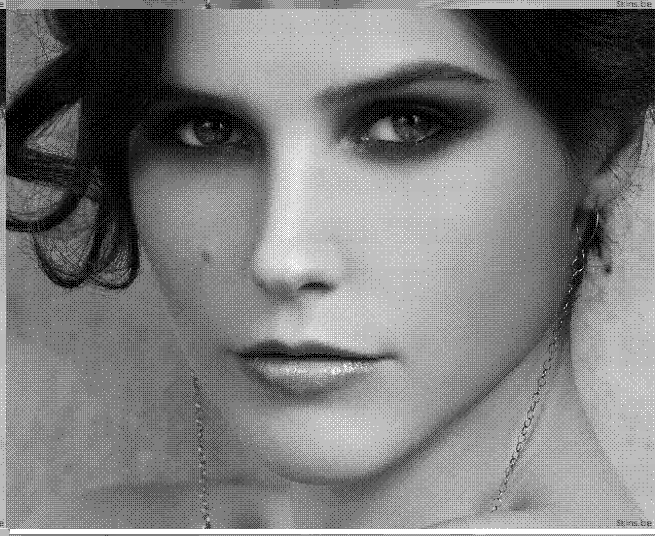
Figura 4 prikazuje polutonirane slike proizvedene sa tehnikom raspršivanje tački čije su indeks matrice veličine 2×2, 4×4, 8×8 i 16×16. Sa slika je jasno da polutoniranje slike omogučava dobru interpretaciju detalja.

Figura 4: Polutone slike proizvedene uz pomoć tehnike Bajer raspršivanja tački čije indeks matrice imaju veličine 2, 4, 8 i 16


Skins.be


Skins.be


Skins.be


Skins.be


WHEATHUMP


WHEATHUMP


WHEATHUMP


WHEATHUMP

2.2.1 Problem broj 4

Napraviti Bayer indeks matrice veličina 2×2, 4×4, 8×8 i 16×16. Napisati funkciju koja će generisati četri različita polutona primjenjujući odgovarajuću dobijenu prag matricu na datu sliku. Isprintati sve četri rezultata u zasebne fajlove.

Za svaku od četri polutona izračunati srednju vrijednost greške kvadratnog korijena (RMSE) i greška srednjeg težinskog kvadratnog korijena (RWMSE) između polutone i orginalne slike.

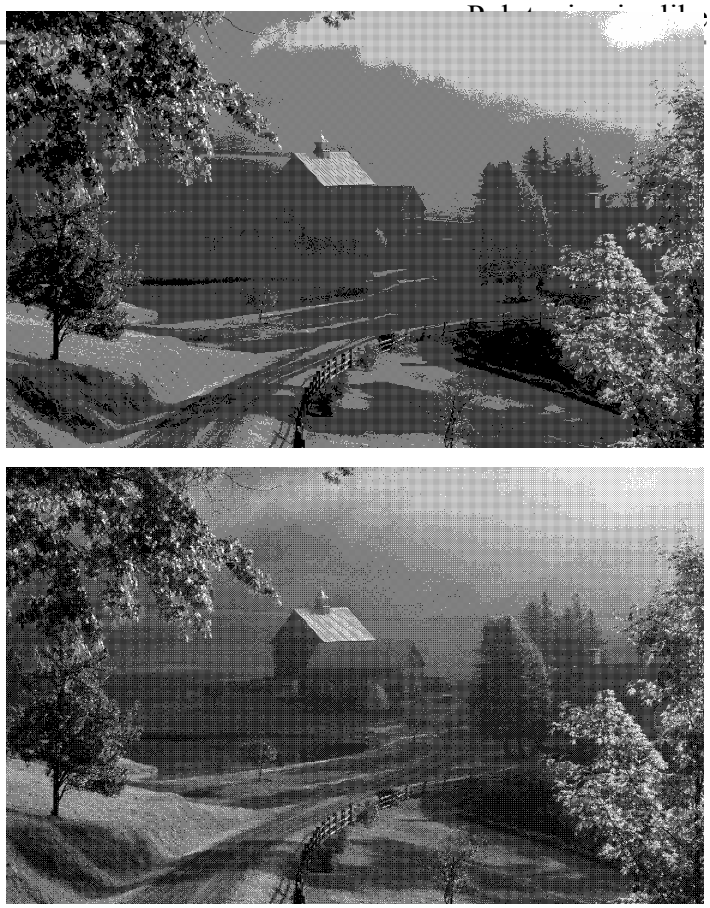
Rješenje

U fajlu dispersed.m se nalazi funkcija koja vrača binarnu sliku dobijenu uz pomoć tehnike raspršivanja tački.

```
function [ orderdith ] = dispersed( sslika, broj )
%% ============================================================
%% ORDERDITH pretvara crnobijelu sliku u binarnu
%% pomoću Bayer tačkastog prikaza
%% Objašnjenje: function [ orderdith ] = ORDERDITH( sslika, broj )
%% sslika = ulazna crnobijela slika (sa grey skalom od 256 nijansi)
%% broj = neki od brojeva 2, 4, 8 ili 16
%% orderdith = izlazna slika
%% ============================================================
[visina, sirina]=size(sslika);

if broj == 2
    in=[1 2; 3 0];
else
    if broj == 4
        in=[1 2; 3 0];
        in=[4*in+1, 4*in+2; 4*in+3, 4*in];
else
    if broj == 8
        in=[1 2; 3 0];
        in=[4*in+1, 4*in+2; 4*in+3, 4*in];
        in=[4*in+1, 4*in+2; 4*in+3, 4*in];
    else
    if broj == 16
        in=[1 2; 3 0];
        in=[4*in+1, 4*in+2; 4*in+3, 4*in];
        in=[4*in+1, 4*in+2; 4*in+3, 4*in];
        in=[4*in+1, 4*in+2; 4*in+3, 4*in];
    end
    end
    end
end

in
i2n=255*((in+0.5)/(broj^2));
orderdith=sslika;

for i=1:1:visina
    for j=1:1:sirina
        m=mod(i,broj)+1;
        n=mod(j,broj)+1;
        if sslika(i,j)>i2n(m,n);
           orderdith(i,j)=255;
        else
           orderdith(i,j)=0;
        end
    end
end

end
```

Kucamo sljedeći kod da dobijemo četri različite slike:

```
>> slika=imread('pr03_cb.tif');
>> dds2_slika=dispersed(slika,2);
in =
     1     2
     3     0
>> dds4_slika=dispersed(slika,4);
in =
     5     9     6    10
    13     1    14     2
     7    11     4     8
    15     3    12     0
>> dds8_slika=dispersed(slika,8);
in =
    21    37    25    41    22    38    26    42
    53     5    57     9    54     6    58    10
    29    45    17    33    30    46    18    34
    61    13    49     1    62    14    50     2
    23    39    27    43    20    36    24    40
    55     7    59    11    52     4    56     8
    31    47    19    35    28    44    16    32
    63    15    51     3    60    12    48     0
```



Slika 6: slike dobijene uz pomoć naše funkcije dispersed čije su veličine indeks matrica redom 2 i 4

```
>> dds16_slika=dispersed(slika,16);
in =

    85   149   101   165    89   153   105   169    86   150   102   166    90   154   106   170
   213    21   229    37   217    25   233    41   214    22   230    38   218    26   234    42
   117   181    69   133   121   185    73   137   118   182    70   134   122   186    74   138
   245    53   197     5   249    57   201     9   246    54   198     6   250    58   202    10
    93   157   109   173    81   145    97   161    94   158   110   174    82   146    98   162
   221    29   237    45   209    17   225    33   222    30   238    46   210    18   226    34
   125   189    77   141   113   177    65   129   126   190    78   142   114   178    66   130
   253    61   205    13   241    49   193     1   254    62   206    14   242    50   194     2
    87   151   103   167    91   155   107   171    84   148   100   164    88   152   104   168
   215    23   231    39   219    27   235    43   212    20   228    36   216    24   232    40
   119   183    71   135   123   187    75   139   116   180    68   132   120   184    72   136
   247    55   199     7   251    59   203    11   244    52   196     4   248    56   200     8
    95   159   111   175    83   147    99   163    92   156   108   172    80   144    96   160
   223    31   239    47   211    19   227    35   220    28   236    44   208    16   224    32
   127   191    79   143   115   179    67   131   124   188    76   140   112   176    64   128
   255    63   207    15   243    51   195     3   252    60   204    12   240    48   192     0
```

```
>> imwrite(dds2_slika,'pr03_dds2.tif','tif')
>> imwrite(dds4_slika,'pr03_dds4.tif','tif')
>> imwrite(dds8_slika,'pr03_dds8.tif','tif')
>> imwrite(dds16_slika,'pr03_dds16.tif','tif')
```

Izračunajmo RMSE za svaku sliku:

```
>> rmse2_slika=rmse(slika, dds2_slika)
rmse2_slika =  110.9580
```

```
>> rmse4_slika=rmse(slika, dds4_slika)
rmse4_slika =  110.1221
>> rmse8_slika=rmse(slika, dds8_slika)
rmse8_slika =  110.1276
>> rmse16_slika=rmse(slika, dds16_slika)
rmse16_slika =  109.8590
```

Izračunajmo još RWMSE za svaku sliku:

```
>> gausovFilter
suma =
   1.0000
c =
   0.0814
gaus =
   0.0009   0.0032   0.0067   0.0086   0.0067   0.0032   0.0009
   0.0032   0.0110   0.0233   0.0300   0.0233   0.0110   0.0032
   0.0067   0.0233   0.0494   0.0634   0.0494   0.0233   0.0067
   0.0086   0.0300   0.0634   0.0814   0.0634   0.0300   0.0086
   0.0067   0.0233   0.0494   0.0634   0.0494   0.0233   0.0067
   0.0032   0.0110   0.0233   0.0300   0.0233   0.0110   0.0032
   0.0009   0.0032   0.0067   0.0086   0.0067   0.0032   0.0009
```

Slika 7: slike dobijene uz pomoć naše funkcije dispersed čije su veličine indeks matrica redom 8 i 16

```
>> f2=primjeniFilter(dds2_slika, gaus);
>> f4=primjeniFilter(dds4_slika, gaus);
>> f8=primjeniFilter(dds8_slika, gaus);
>> f16=primjeniFilter(dds16_slika, gaus);

>> rwmse2_slika=rmse(slika, f2)
rwmse2_slika =
  20.9364

>> rwmse4_slika=rmse(slika, f4)
rwmse4_slika =
  17.7922

>> rwmse8_slika=rmse(slika, f8)
rwmse8_slika =
  17.9411

>> rwmse16_slika=rmse(slika, f16)
rwmse16_slika =
  17.9462
```

2.3 Zadatak za vježbu

Primjeniti tehniku raspršivanja tački na neku ulaznu sliku uz pomoć Void i Cluster indeks matrice veličine 128×128. Izračunati RMSE i RWMSE za dobijenu sliku.

III Greška širenja

Još jedna metoda za polutoniranje je error diffusion (greška širenja). U ovom slučaju pikseli se kvantiziraju u određenom redu (raster uređenje se često koristi. Objašnjenje: Raster uređenje slike obilazi piksele sa lijeve na desnu stranu, odozgo prema dole. Ovo je slično uređenju koje koristi CRT dok skenira elektronske zrake preko ekrana), i ostatak greški kvantizacije za trenutni piksel se množi (širi) naprijed na sljedeći nekvantizirani piksel. Ovo drži sveukupni intenzitet izlazne binarne slike bliže intenzitetu ulazne sive skale.

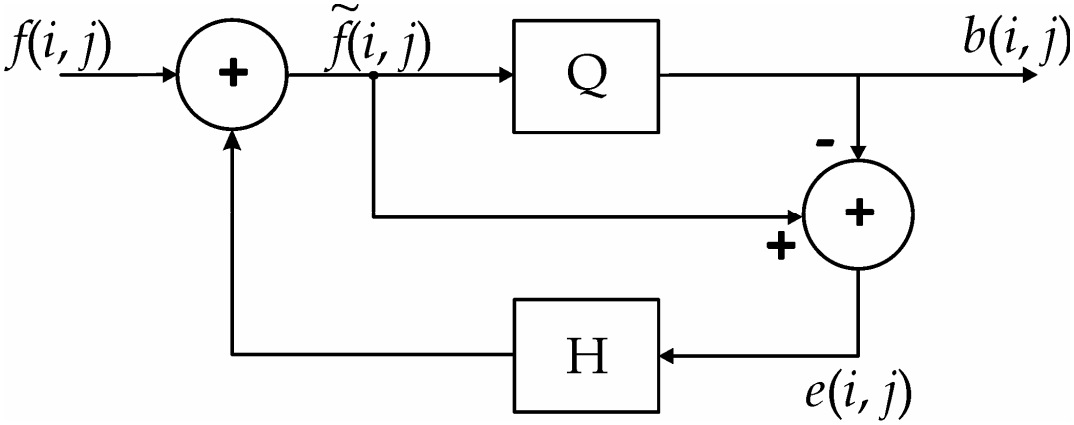


Figura 5: Blok dijagram metode greške širenja

Figura 5 je blok dijagram koji ilustrira metodu greške širenja. Trenutni ulazni piksel $f(i, j)$ je promjenjen kao posljedica ranije već dobijene greške kvantizacije, što daje promjenjen ulaz $\tilde{f}(i, j)$. Piksel se tad kvantizira u binarnu vrijednost sa Q, koji koristi neki prag T. Greška $e(i, j)$ je definisana sa

$$e(i, j) = \tilde{f}(i, j) - b(i, j) \qquad (9)$$

gdje je $b(i, j)$ kvantizirana binarna slika.

Greška $e(i, j)$ kvantizacije trenutnog piksela se širi na "sljedeći(e)" piksel(e) na neki dvodimenzionalni način uz pomoć filtera $h(i, j)$, poznat kao diffusion filter (filter širenja). Ovaj proces mijenjanja ulazni piksel zajedno sa greškom koja se pojavila, i može biti predstavljena pomoću sljedeće rekursivne relacije:

$$\tilde{f}(i, j) = f(i, j) + \sum_{k, l \in S} h(k, l)\, e(i-k, j-l) \qquad (10)$$

3.1 Floyd Steinberg Error Diffusion (1976)

Najpopularniji metod greške širenja su predložili Floyd i Steinberg [2], koristi filter raspršivanja pokazanoj na Figuri 6. S obzirom da je suma koeficijenata filtera jednaka 1, lokalna prosječna vrijednost kvantizirane slike je jednaka lokalnoj prosječnoj vrijednosti sive slike. Figura 7 pokazuje polutonirane slike proizvedene pomoću Floyd i Steinberg greške širenja. Ako uporedimo dobijene rezultate sa uređenim polutoniranim slikama, može se primjetiti da metoda greške širenja ima bolje osobine za kontrast slike. Kako god, može se vidjeti sa Figure 7 da greška širenja teži da napravi "prošireni" artefekt, poznat pod imenom crv uzorak.

	●	7/16
3/16	5/16	1/16

Figura 6: Funkcija širenja tačke, filtera greške širenja predložen od Floyda i Steinberga

Figura 7: Orginalne i polutonirane slike proizvedene sa Floyd i Steinberg metodom greške širenja

3.1.1 Problem broj 5

Primjeniti tehniku greške širenja na neku ulaznu sliku. Za prag koristiti vrijednost 127 i filter širenja predstavljen u Figuri (6).

Izračunati grešku srednje vrijednosti kvadratnog korijena (RMSE) koji je definisana sa (2) na strani 6. Također izračunati grešku srednjeg težinskog kvadratnog korijena (RWMSE) definisanu sa (6) na strani 14.

Rješenje:

U fajl FloStei.m ćemo snimiti funkciju koja primjenjuje filter širenja predstavljen u Figuri (5).

```
function [diffusion] = FloStei(sslika)
%% ========================================
%% FLOSTEI pretvara sivu sliku u binarnu
%% koristeći Floyd Steinberg tehniku greške širenja
%% Objašnjenje: function [diffusion] = FloStei(sslika)
%% sslika = ulazna siva slika
%% diffusion = dobijena binarna slika
%% ========================================

[visina, sirina]=size(sslika);
diffusion=double(sslika);
b=double(sslika);
e=double(sslika);
for i=1:1:(visina-1)
    for j=1:1:sirina
% prvi ljevi pikseli ============================
        if(i==1)
            if (j==sirina)
                if(diffusion(i,j)>127)
                    b(i,j)=255;
                else
                    b(i,j)=0;
                end
                e(i,j)=diffusion(i,j)-b(i,j);
                diffusion(i+1,j-1)=diffusion(i+1,j-1)+(3/16)*e(i,j);
                diffusion(i+1,j)=diffusion(i+1,j)+(5/16)*e(i,j);
                diffusion(i,j)=b(i,j);
            else
            if(diffusion(i,j)>127)
                b(i,j)=255;
            else
                b(i,j)=0;
            end
            e(i,j)=diffusion(i,j)-b(i,j);
            diffusion(i,j+1)=diffusion(i,j+1)+((7/16)*e(i,j));
            diffusion(i+1,j)=diffusion(i+1,j)+(5/16)*e(i,j);
            diffusion(i+1,j+1)=diffusion(i+1,j+1)+(1/16)*e(i,j);
            diffusion(i,j)=b(i,j);
            end
% ============================================
        else
% pikseli koji se nalaze na kraju sirine ========
             if(j==sirina)
                if(diffusion(i,j)>127)
                    b(i,j)=255;
                else
                    b(i,j)=0;
                end
                e(i,j)=diffusion(i,j)-b(i,j);
                diffusion(i+1,j-1)=diffusion(i+1,j-1)+(3/16)*e(i,j);
                diffusion(i+1,j)=diffusion(i+1,j)+(5/16)*e(i,j);
                diffusion(i,j)=b(i,j);
```

```
% ================================================
            else if (j==1)
                    if(diffusion(i,j)>127)
                        b(i,j)=255;
                    else
                        b(i,j)=0;
                    end
                    e(i,j)=diffusion(i,j)-b(i,j);
                    diffusion(i,j+1)=diffusion(i,j+1)+(7/16)*e(i,j);
                    diffusion(i+1,j)=diffusion(i+1,j)+(5/16)*e(i,j);
                    diffusion(i+1,j+1)=diffusion(i+1,j+1)+(1/16)*e(i,j);
                    diffusion(i,j)=b(i,j);
                else
% Pikseli koji se nalaze u sredini ==============
        if(diffusion(i,j)>127)
            b(i,j)=255;
        else
            b(i,j)=0;
        end
        e(i,j)=diffusion(i,j)-b(i,j);
        diffusion(i,j+1)=diffusion(i,j+1)+(7/16)*e(i,j);
        diffusion(i+1,j-1)=diffusion(i+1,j-1)+(3/16)*e(i,j);
        diffusion(i+1,j)=diffusion(i+1,j)+(5/16)*e(i,j);
        diffusion(i+1,j+1)=diffusion(i+1,j+1)+(1/16)*e(i,j);
        diffusion(i,j)=b(i,j);
% ========================================================
                end
            end
        end
    end
end
% za zadnji red piksela
i=visina;
for j=1:1:(sirina-1)
    e(i,j)=diffusion(i,j)-b(i,j);
    diffusion(i,j+1)=diffusion(i,j+1)+(7/16)*e(i,j);
    diffusion(i,j)=b(i,j);
end
% ====================
end
```

Ako sad ukucamo:

```
>> slika=imread('pr03_cb.tif');
>> fs_slika=FloStei(slika);
>> imwrite(fs_slika,'pr03_cb_fs.tif','tiff')
```

kao rezultat imamo sliku predstavljenu desno od orginalne slike:

Slika 8:
(a) orginalna siva slika
(b) slika dobijena uz pomoć naše funkcije FloStei

Izračunajmo grešku srednje vrijednosti kvadratnog korijena (RMSE):

```
>> rmse_s=rmse(slika, fs_slika)
rmse_s =  109.0383
```

Izračunajmo grešku srednjeg težinskog kvadratnog korijena (RWMSE):

```
>> gausovFilter
suma =
    1.0000
c =
    0.0814
gaus =
    0.0009    0.0032    0.0067    0.0086    0.0067    0.0032    0.0009
    0.0032    0.0110    0.0233    0.0300    0.0233    0.0110    0.0032
    0.0067    0.0233    0.0494    0.0634    0.0494    0.0233    0.0067
    0.0086    0.0300    0.0634    0.0814    0.0634    0.0300    0.0086
    0.0067    0.0233    0.0494    0.0634    0.0494    0.0233    0.0067
    0.0032    0.0110    0.0233    0.0300    0.0233    0.0110    0.0032
    0.0009    0.0032    0.0067    0.0086    0.0067    0.0032    0.0009
>> slika_gaus=primjeniFilter(fs_slika,gaus);
>> rwmse_s=rmse(slika, slika_gaus)
rwmse_s =
   17.6113
```

Polutonirana slika na koju primjenimo Gausov filter (definisan sa (7) na strani 17) izgleda ovako:

Slika 9: Slika dobijena kao rezultat primjene Gausovog filtera na polutoniranu sliku dobijenu uz pomoć tehnike greške širenja

3.2 Zadatak za vježbu

Često korišten metod greške širenja, pored prethodnog, su predložili Jarvis, Judice i Ninke (1976) i on koristi filter raspršivanja pokazanoj u Figuri 8.

			7/48	5/48
3/48	5/48	7/48	5/48	3/48
1/48	3/48	5/48	3/48	1/48

Figura 8: Funkcija širenja tačke, filtera greške širenja predložen od Jarvis-a, Judice-a i Ninke-a

Primjeniti tehniku greške širenja na neku ulaznu sliku. Za prag koristiti vrijednost 127 i filter širenja predstavljen u Figuri (7).

Izračunati grešku srednje vrijednosti kvadratnog korijena (RMSE) koji je definisana sa (2) na strani 6. Također izračunati grešku srednjeg težinskog kvadratnog korijena (RWMSE) definisanu sa (6) na strani 14.

Literatura

[1] B. E. Bayer, "An optimum method for two-level rendition of continuos-tone pictures," IEEE International Conference on Communications, vol. 1, June 11-13 1973, str. 11-15

[2] R.W. Floyd i L. Steinberg, "An adaptive algorithm for spatial greyscale," Journal of the Society for Information Display, vol. 17, no 2, pp. 75-77, 1976.

[3] https://engineering.purdue.edu/~bouman/ece637/
https://engineering.purdue.edu/~bouman/ece637/lectures07/
Video predavanja pod rednim brojem 36, 37 i 38
https://engineering.purdue.edu/~bouman/grad-labs/lab7/
https://engineering.purdue.edu/~bouman/ece637/notes/

[4] R. C. Gonzales, R. E. Woods, S. L. Eddins, "Digital Image Processing Using Matlab", Pearson 2004, str. 108-143