SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

VARAŽDIN

Sandra Sačarić

Ordered Dithering algoritam

Varaždin, 28.1.2024.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

VARAŽDIN

Sandra Sačarić

Ordered Dithering algoritam

Studij: Informacijski i poslovni sustavi

Mentor: Prof. Dr. Sc. Alen Lovrenčić

Varaždin, 28.1.2024.

1. Sadržaj

1. Uvod

U digitalnoj obradi slika, postizanje optimalne kvalitete slike često predstavlja izazov, posebno kada se radi o smanjenju broja boja ili konverziji slika u crno-bijeli format. Ordered dithering algoritam predstavlja jednostavan, ali učinkovit pristup ovom problemu. Ovaj algoritam koristi *threshold* matricu kako bi svaki piksel slike usporedio s određenim pragom, na temelju čega se donosi odluka o boji piksela u rezultirajućoj crno-bijeloj slici. Princip algoritma i ciljani efekt detaljnije će se objasniti u daljnjim stavkama ovog seminarskog rada. Objašnjenje ideje iza algoritma proizlazi iz stranice na Wikipediji [1].

Ovaj seminarski rad fokusirat će se na implementaciju te izračunu složenosti Ordered Dithering algoritma s posebnim naglaskom na konverziju u crno-bijelo sliku, ali će se pokušati obraditi i mogućnost očuvanja boje slike nakon konverzije. Ovaj algoritam ima široku primjenu u različitim područjima, uključujući tiskanje, web dizajn, tehnologiju prikaza i druge. Implementacija će se temeljiti na programskom jeziku C++, pružajući jednostavnu i učinkovitu demonstraciju algoritma.

U nastavku rada, analizirat će se teorijski aspekti Ordered Dithering algoritma, uključujući i princip rada. Detaljno će se razmotriti dvije implementacije algoritma – rezultat algoritma u crno-bijeloj formi te pokušaj održavanje boje nakon konverzije, koristeći različite veličine threshold matrica. Prikazat će se rezultati primjene algoritma na par primitivnih slika samostalne izrade (boje poredane u 8x8 polja), s naglaskom na vizualnu analizu i usporedbu s originalnim slikama.

U konačnici, ovaj rad ima za cilj pružiti čitatelju uvid u primjenu Ordered Dithering algoritma te analizirati kako različite veličine threshold matrica utječu na kvalitetu rezultirajućih slika.

2. Princip rada algoritma

Ordered Dithering je tehnika obrade slika koja se često koristi za smanjenje broja boja u digitalnim slikama, posebno pri konverziji u crno-bijeli format. Ovaj algoritam koristi threshold matricu kako bi svaki piksel slike usporedio s odgovarajućim pragom. Na temelju rezultata usporedbe, donosi se odluka o boji piksela u rezultirajućoj slici.

Ulazni podaci:

Ulazni podaci uključuju originalnu sliku. Kod primjera gdje su rezultat crno-bijele vrijednosti, slika će biti prikazana u formi 2D vektora (matricu vrijednosti piksela slike), koja se želi konvertirati u crno-bijeli format, dok će se kod primjera s očuvanjem boja željena slika prikazati kao 3D vektor (svako polje imat će tri dimenzije, a svaka dimenzija predstavlja jednu boju prema RGB sistemu).

Generiranje threshold matrice:

Algoritam počinje generiranjem threshold matrice. Veličina ove matrice određuje se unaprijed, a najčešće se koriste matrice dimenzija 2x2, 4x4, ili 8x8.

Iteracija kroz sliku:

1. **Crno-bijela slika:**

Svaki piksel u originalnoj slici se uspoređuje s odgovarajućim elementom threshold matrice. Ukoliko je vrijednost **piksela** veća od vrijednosti elementa **threshold** matrice, piksel se postavlja na vrijednost koja predstavlja crnu boju – vrijednost 0. Inače, piksel se postavlja na maksimalnu vrijednost boje – vrijednost 255, a predstavlja bijelu boju.

Naravno, kako je ovdje riječ o konverziji originalne slike u crno-bijelu varijantu, iz tog razloga su postavljene samo dvije rezultirajuće vrijednosti (već spomenute, 0 i 255, odnosno crna i bijela), ali s obzirom da dithering algoritam može služiti za konverziju slike u boji, moguće je definirati i druge slučaje.

2. **Slika u boji:**

Ako bi se htjelo raditi na konverziji slika gdje se boje nastoje očuvati, morale bi se definirati razine boje crvene, plave, i zelene (prema RGB sistemu). Svaki bi piksel slike bio trodimenzionalno polje s vrijednostima svake navedene boje, čije bi se vrijednosti zatim ispitivale (uspoređivale sa threshold matricom). To će se postići na način da se slika definira kao 3D vektor.

Kako je poanta dithering algoritma da rezultirajuća slika bude prikaz boja koristeći samo vrijednosti 0 ili 255, ugniježđene petlje prolaze kroz svaki piksel u slici, gdje se svaki piksel predstavlja strukturom „Boja“ koja sadrži crvenu, zelenu i plavu komponentu. Pristupanje pojedinim kanalima boje (crveni, zeleni, plavi) zasebno se podvrgava postupku algoritma.

Na primjer, ukoliko bi se kao ulaz postavili sliku roze boje, kako je roza zapravo kombinacija crvene i plave boje (obje u određenoj mjeri), kao rezultat bi se dobila polja s vrijednostima {255, 0, 255}.

Primjena na cijelu sliku:

Postupak se iterativno ponavlja za svaki piksel u slici, osiguravajući da svaki piksel dobije boju ovisno o rezultatu usporedbe s pragom iz threshold matrice.

Primjena različitih veličina threshold matrica:

Odabir različitih veličina threshold matrica utječe na rezultat. Manje matrice mogu rezultirati grubljim teksturama, dok veće matrice mogu proizvesti glađe prijelaze između različitih boja.

Prednosti:

- Jednostavna implementacija.
- Učinkovita u smanjenju broja boja s minimalnim gubitkom detalja.

Ograničenja:

- Može uzrokovati "mrežasti" efekt na slikama s finim detaljima.
- Rezultat ovisi o odabiru veličine threshold matrice.

3. Implementacije algoritma

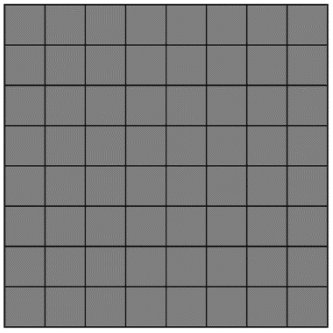
U nastavku rada, prikazat će se implementacija Ordered Dithering algoritma u programskom jeziku C++ i analizirati rezultate primjene algoritma.

3.1. Implementacija algoritma – rezultat kao isključivo crno-bijela slika

```
1  #include <iostream>
2  #include <vector>

3  // funkcije preko koje ce se odrediti dimenzija threshold matrice i unositi elementi
4  std::vector<std::vector<int>> stvoriMatricu(int dim) {
5      std::vector<std::vector<int>> matrica(dim, std::vector<int>(dim, 0));
6      std::cout << "Unos elementa threshold matrice " << dim << "x" << dim << " dimenzija:\n";

7      for (int i = 0; i < dim; ++i) {
8          for (int j = 0; j < dim; ++j) {
9              std::cout << "Element pozicije (" << i << ", " << j << "): ";
10             std::cin >> matrica[i][j];
11         } // for
12     } // for

13     return matrica;
14 } // funkcija

15 // funkcija preko koje ce se izvesti algoritam na primjeru crno-bijele slike
16 void orderedDithering(std::vector<std::vector<int>>& slika, const std::vector<std::vector<int>>&
   thresholdMatrica) {
17     int visina = slika.size();
18     int sirina = slika[0].size();
19     int matricaDim = thresholdMatrica.size();
20
21     for (int y = 0; y < visina; ++y) {
22         for (int x = 0; x < sirina; ++x) {
23             int threshold = thresholdMatrica[y % matricaDim][x % matricaDim];
24             int novaVrijednost = (slika[y][x] > threshold) ? 255 : 0;
25             slika[y][x] = novaVrijednost;
26         } // for
27     } // for
28 } // funkcija
```

```
29  int main() {
30      // primjer crno-bijele slike u formi 2D vektora
31      // prikazivat ce se slika iskljucivo jedne boje kako bi se uspjesnije svhatio princip rada algoritma
32      // 128 oznacava sredisnju vrijednost izmedju 255 i 0, odnosno sivu boju (izmedju bijele i crne)
33      std::vector<std::vector<int>> slika = {
34          {128, 128, 128, 128, 128, 128, 128, 128},
35          {128, 128, 128, 128, 128, 128, 128, 128},
36          {128, 128, 128, 128, 128, 128, 128, 128},
37          {128, 128, 128, 128, 128, 128, 128, 128},
38          {128, 128, 128, 128, 128, 128, 128, 128},
39          {128, 128, 128, 128, 128, 128, 128, 128},
40          {128, 128, 128, 128, 128, 128, 128, 128},
41          {128, 128, 128, 128, 128, 128, 128, 128}
42        };
43
44      // odredjivanje dimenzija matrice preko koje se radi usporedba vrijednosti piksela slike
45      //preporucene dimezije matrica su 2x2, 4x4, ili 8x8
46      int dim;
47      std::cout << "Dimenzija threshold matrice (npr., 2, 4, 8): ";
48      std::cin >> dim;
49
50      // stvaranje matrice
51      std::vector<std::vector<int>> thresholdMatrica = stvoriMatricu(dim);
52
53      // izvedba algoritma
54      orderedDithering(slika, thresholdMatrica);
55
56      // prikaz slike
57      for (const auto& redak : slika) {
58          for (int pixel : redak) {
59              std::cout << pixel << " ";
60          } // for
61          std::cout << std::endl;
62      } // for
63
64      return 0;
65  } // main
```

Slika koja će služiti kao primjer bit će 8x8 matrica gdje svako polje predstavlja vrijednost 128:



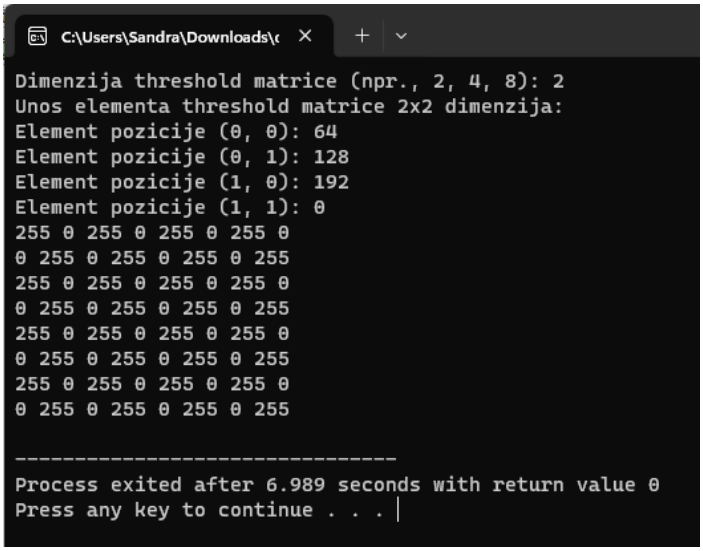
Slika 1 Primjer slike za crno-bijelu konverziju (autorski rad)

Sada kada imamo sliku, potrebno je odrediti dimenziju threshold matrice te njene vrijednosti. U svrhe lakšeg razumijevanja, krenut će se sa dimenzijom matrice 2x2, a početne vrijednosti threshold matrice će biti {{64, 128},{192, 0}}.

Sada će se nastojati objasniti princip rada algoritma. Vanjska petlja prolazi kroz redove slike (**visina**), a unutarnja petlja prolazi kroz stupce slike (**širina**). Za svaki piksel slike (**slika[y][x]**), uzima se prag (**threshold**) iz odgovarajuće pozicije u matrici praga.

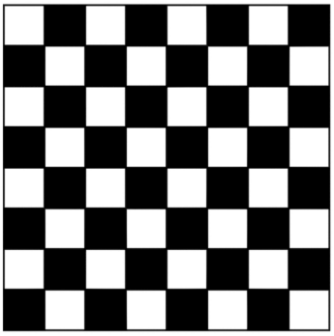
U ovom koraku koristi se mod operator kako bi se omogućila ponovna upotreba pragova iz matrice praga kada se dođe do kraja reda ili stupca. Nova vrijednost piksela (**novaVrijednost**) postavlja se na 255 ako je vrijednost piksela veća od praga, a inače se postavlja na 0.

Konkretno u ovom primjeru, uzima se element prvog reda u prvom stupcu zapisa slike ($zS_{1,1} = 128$) te se uspoređuje sa 64, prvim elementom unutar threshold matrice, skraćeno $tM_{1,1}$. Kako je 128 veći od 64, bilježi se 255. Zatim se unutarnjom petljom ide dalje do drugog elementa $zS_{1,2} = 128$ koji je jednake vrijednosti kao i prvi, ali ga ovog puta uspoređujemo s $tM_{1,2}$ koji je također 128 te od kojeg nije strogo veći (jednake su vrijednosti) pa se zapisuje 0. Tako redom do kraja prvog **reda slike** vrijednosti naizmjence uspoređujemo isključivo s vrijednostima {64, 128} threshold matrice. Kada se prelazi na drugi redak slike, tada se tek kreće s usporedbom vrijednosti na donjem redu threshold matrice $tM_{2,1}$ i $tM_{2,2}$, odnosno vrijednosti {192, 0}. Element drugog reda u prvom stupcu zapisa slike $zS_{2,1} = 128$ nije veći od 192, stoga zapisujemo 0. Nadalje, $sZ_{2,2} = 128$ je veći od 0 pa zapisujemo 255. Postupak se ponavlja do zadnjeg stupca u zadnjem redu.



Slika 2 Ispis programa koristeći prvu implementaciju (autorski rad)

Gore je ispis programa koristeći parametre koji su navedeni u objašnjenju. Ako se ponovi da 255 predstavlja bijelu, a 0 crnu boju, nakon izvedbe rada algoritma rezultat je sljedeći:



Slika 3 Prikaz rezultat kod ispisa crno-bijelih vrijednosti (autorski rad)

Naravno, na prvi pogled se ne čini kao da je algoritam uspješan, ali to je zato što je primjer dosta jednostavan u svrhe lakšeg shvaćanja. Kada bi slika bila veća od samo 8x8 polja, postigla bi se i složenija konverzija. Dokaz tome, čak i bez mijenjanja primjera, leži u činjenici da se prikaz rezultata promjeni i krene izgledati nalik originalnoj slici ako dovoljan broj puta pritisnemo zoom out.

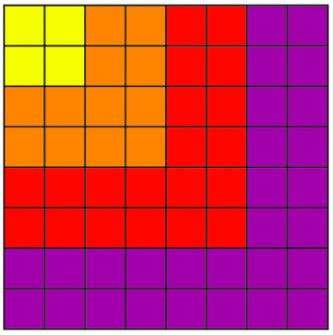
Ideja iza primjera i zadani brojevi, kao i objašnjenje postupka algoritma proizašla je iz videa s Youtube kanala Computerphile [2].

3.2. Implementacija algoritma – rezultat u boji

```
#include <iostream>
#include <vector>

// struktura koja predstavlja boje RGB sustava
struct Boja {
  int R, G, B;
  Boja(int r, int g, int b) : R(r), G(g), B(b) {}
};

// funkcija algoritma
void orderedDithering(std::vector<std::vector<Boja>>& slika) {
// definiranje threshold matrice
  int thresholdMatrica[4][4];

  int visina = slika.size();
  int sirina = slika[0].size();

// izvedba algoritma za svaki pojedinacni kanal
  for (int y = 0; y < visina; ++y) {
    for (int x = 0; x < sirina; ++x) {
      Boja& pixel = slika[y][x];
      pixel.R = (pixel.R * 15 / 255 > thresholdMatrica[y % 4][x % 4]) ? 255 : 0;
      pixel.G = (pixel.G * 15 / 255 > thresholdMatrica[y % 4][x % 4]) ? 255 : 0;
      pixel.B = (pixel.B * 15 / 255 > thresholdMatrica[y % 4][x % 4]) ? 255 : 0;
    } // for
  } // for
} // funkcija

int main() {
  // fiksni primjer 8x8 slike kao prikaz 3D vektora
  // svako polje je trodimenzionalni prikaz {R, G, B} vrijednosti
  std::vector<std::vector<Boja>> slika = {
    {{245, 255, 0}, {245, 255, 0}, {255, 130, 0}, {255, 130, 0}, {255, 0, 0}, {255, 0, 0}, {160, 0, 170}, {160, 0, 170}},
    {{245, 255, 0}, {245, 255, 0}, {255, 130, 0}, {255, 130, 0}, {255, 0, 0}, {255, 0, 0}, {160, 0, 170}, {160, 0, 170}},
    {{255, 130, 0}, {255, 130, 0}, {255, 130, 0}, {255, 130, 0}, {255, 0, 0}, {255, 0, 0}, {160, 0, 170}, {160, 0, 170}},
    {{255, 130, 0}, {255, 130, 0}, {255, 130, 0}, {255, 130, 0}, {255, 0, 0}, {255, 0, 0}, {160, 0, 170}, {160, 0, 170}},
    {{255, 0, 0}, {255, 0, 0}, {255, 0, 0}, {255, 0, 0}, {255, 0, 0}, {255, 0, 0}, {160, 0, 170}, {160, 0, 170}},
    {{255, 0, 0}, {255, 0, 0}, {255, 0, 0}, {255, 0, 0}, {255, 0, 0}, {255, 0, 0}, {160, 0, 170}, {160, 0, 170}},
    {{160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}},
    {{160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}, {160, 0, 170}}
};

  // izvedba algoritma
  orderedDithering(slika);
```

```
39    // prikaz rezultata
40    for (const auto& redak : slika) {
41      for (const auto& pixel : redak) {
42        std::cout << "(" << pixel.R << "," << pixel.G << "," << pixel.B << ") ";
43      } // for
44      std::cout << std::endl;
45    } // for

46  return 0;
47  } // main
```

Slika (8x8) koja je izrađena za primjer i koja je pikazana u kodu izgleda ovako:

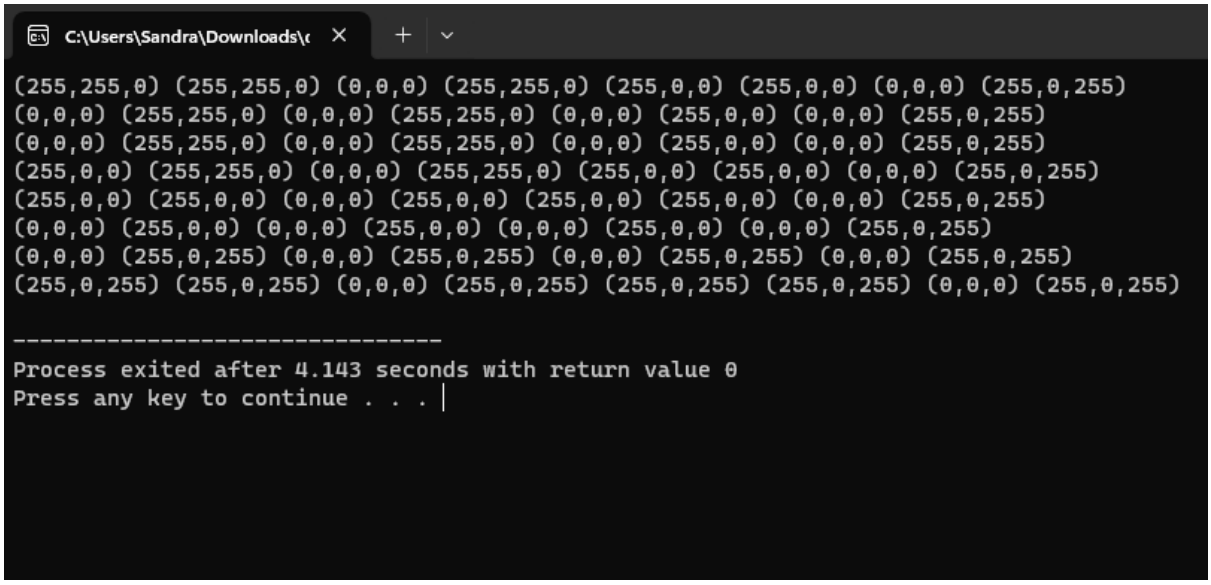


Slika 4 Primjer slike za konverziju u boji (autorski rad)

Kako je već objašnjen princip po kojem algoritam radi, fokus će se staviti na razliku između prve implementacije i trenutne.

Ugniježđene petlje prolaze kroz svaki piksel u slici, gdje se svaki piksel predstavlja strukturom „Boja“ koja sadrži crvenu, zelenu i plavu komponentu.

Nadalje, svaki kanal boje (R, G, B) zasebno se podvrgava postupku ordered ditheringa, odnosno uspoređuje se sa zadanom threshold matricom. Za svaki kanal, intenzitet boje se skalira na raspon od 0 do 15 koristeći threshold matricu. Ako je skalirani intenzitet veći od odgovarajuće vrijednosti u threshold matrici, postavlja se na maksimalnu vrijednost 255; inače se postavlja na vrijednost 0. Time se postiže efekt ordered ditheringa za svaki kanal boje zasebno.



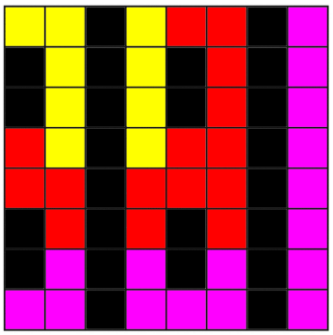
Slika 5 Ispis programa koristeći drugu implementaciju (autorski rad)

Nakon primjene naručenog diteringa na sve piksele, rezultati se ispisuju, prikazujući RGB vrijednosti svakog piksela.

Konkretno na primjeru slike, prvo se ispituje $zS_{1,1}$ čiji RGB zapis iznosi $\{245, 255, 0\}$. To je zapis određene nijanse žute boje. Ako se promotri originalnu sliku (ili njen zapis u kodu), vidjet će se da postoje četiri polja te iste nijanse žute boje. S obzirom da kod ove implementacije kao rezultat ispitivanja mogu biti isključivo krajnje vrijednosti (255 ili 0), prilikom izvedbe algoritma kao rezultat će se pojaviti RGB zapis $\{255, 255, 0\}$ u tri od ta spomenuta četiri polja. Budući da to ipak nije jednak zapis onome u originalnoj slici, odredit će se da jedan od četiri polja ima postavljen zapis $\{0, 0, 0\}$.

Neka se prijeđe na nijansu narančaste boje čiji je zapis u originalnoj slici $\{255, 130, 0\}$. Narančasta boja nalazi se između crvene i žute pa se prema tome kao rezultat ordered ditheringa, osim $\{0, 0, 0\}$ koji služi kao prijelazna vrijednost među bojama, mogu se pronaći samo kombinacije zapisa $\{255, 255, 0\}$ koji predstavlja žutu te $\{255, 0, 0\}$ koji predstavlja crvenu.

Isti princip vrijedi i za zadnju boju (ljubičastu) pa nema potrebe dodatno pojašnjavati postupak. Rezultat algoritma nad gore navedenom slikom u ovoj implementaciji izgledao bi ovako:



Slika 6 Prikaz rezultat sa očuvanim bojama (autorski rad)

Rezultat nije pretjerano ugodan oku, ali to je zato što kod ove implementacije nisu dodatno omogućeni prijelazi boja i jer je ordered dithering algoritam **primarno namjenjen za konverziju crno-bijelih slika**. Druga implementacija je zapravo samo nadobudni pothvat autora ovog seminara u postizanju više razine funkcionalnosti.

4. Izračun složenosti algoritma

Kako je već bilo spomenuto da je ovaj algoritam primarno namjenjen konverziji crno-bijelih slika i s obzirom da su obje implementacije u ovom seminaru relativno slične, konkretan će se izračun napraviti koristeći kôd prve implementacije.

Za početak će se prikazati izračun kompleksnosti funkcije stvoriMatricu.

```
std::vector<std::vector<int>> stvoriMatricu(int dim) {
   std::vector<std::vector<int>> matrica(dim, std::vector<int>(dim, 0));

   std::cout << "Unos elementa threshold matrice " << dim << "x" << dim << " dimenzija:\n";  // c1

   for (int i = 0; i < dim; ++i) {                                          // ×(n) ≤ c2
      for (int j = 0; j < dim; ++j) {                                       // ×(n) ≤ c3
         std::cout << "Element pozicije (" << i << ", " << j << "): ";     // c4
         std::cin >> matrica[i][j];                                         // c5
      }
   }
   return matrica;                                                          // c6
}
```

Kako dimenzije threshold matrice moraju biti jednake, kroz petlje se prolazi $dim \times dim$ broj puta. U svrhe lakšeg zapisa, umjesto dim će se pisati n.

Za prvu petlju, sve tri klauzule su konstatne složenosti, a zapisat će se kao a_1, a_2, a_3. U prvom koraku petlje će se izvoditi a_1 te a_2 pa je zato $b_1 = a_1 + a_2$, dok se u ostalim koracima izvode treća, nakon koje ide druga klauzula, stoga vrijedi $b_2 = a_3 + a_2$. Stoga vrijedi da je $c_2 = \max\{b_1, b_2\}$ pa je iz tog razloga složenost kontrole prve petlje manja ili jednaka c_2. Isti princip bi se primjenio za provjeru složenosti druge petlje.

Naredbe u jednom koraku prve for petlje su složenosti $d_1 = c_2$, a budući da petlja ima n koraka, uzima se ukupna složenost $d_1 \cdot n$. Isti princip vrijedi i za drugu petlju, samo što $d_2 = c_3 + c_4 + c_5$.

Sada je ukupna složenost funkcije:

$$\begin{aligned} T_{max}^{matrica}(n) &\leq c_1 + (d_1 \cdot n\,(d_2 \cdot n)) \\ &= c_1 + d_1 n + d_2 n + d_1 d_2 + n^2 \\ T_{max}^{matrica}(n) &= O(n^2) \end{aligned}$$

Kako je funkcija orderedDithering (postupak samog algoritma) gotovo identičan, uz iznimku da sada slika ne mora imati iste dimenzije, složenost može također iznositi $O(n^2)$, odnosno $O(n \cdot m)$ ako se drži prethodno navedene tvrdnje pa se visina slike gleda kao dimenzija n, a širina kao dimenzija m.

```
void orderedDithering(std::vector<std::vector<int>>& slika, const std::vector<std::vector<int>>&
thresholdMatrica) {
   int visina = slika.size();                                                    // c1
   int sirina = slika[0].size();                                                 // c2
   int matricaDim = thresholdMatrica.size();                                     // c3

   for (int y = 0; y < visina; ++y) {                                            // ×(n) ≤ c4
      for (int x = 0; x < sirina; ++x) {                                         // ×(n) ≤ c5
         int threshold = thresholdMatrica[y % matricaDim][x % matricaDim];       // c6
         int novaVrijednost = (slika[y][x] > threshold) ? 255 : 0;               // c7
         slika[y][x] = novaVrijednost;                                           // c8
      }
   }
}

int dim;                                                                         // 0
   std::cout << "Dimenzija threshold matrice (npr., 2, 4, 8): ";                 // c1
   std::cin >> dim;                                                              // c2

   std::vector<std::vector<int>> thresholdMatrica = stvoriMatricu(dim);          // c3

   orderedDithering(slika, thresholdMatrica);                                    // c4

   for (const auto& redak : slika) {                                             // ×(n) ≤ c5
      for (int pixel : redak) {                                                  // ×(n) ≤ c6
        std::cout << pixel << " ";                                               // c7
      }
      std::cout << std::endl;                                                    // c8
   }

   return 0;                                                                     //c9
}
```

Ispis unutar main funkcije također ima $n \cdot m$ koraka petlje (s obzirom da je potrebno ispisati svaki piksel slike), stoga je kao i prethodne funkcije složenost jednaka $O(n \cdot m)$.

Za svaki izračun složenosti, korištene su prezentacije prof. Lovrenčića s LMS sustava *elf* kao pomoć pri shvaćanju postupka; konkretno prezentacije s predavanja [3].

5. Zaključak

Ordered dithering algoritam je učinkovita tehnika obrade slika koja se koristi za postizanje umjetničkih efekata, simulaciju dodatnih boja, a ponajviše smanjenje količine boja u slici. Ova tehnika se temelji na primjeni unaprijed definiranih matrica preko kojih će se izvoditi usporedba na sliku, čime se stvara dojam dodatnih nijansi i tekstura. Na temelju implementacija ovog algoritma, može se donijeti nekoliko ključnih zaključaka.

Prvo, ordered dithering omogućuje prilagodbu pragova za svaku pojedinu boju na temelju intenziteta boje. Drugo, korištenje threshold matrice doprinosi strukturi ordered ditheringa. Veličina i vrijednosti u matrici mogu se prilagoditi kako bi se postigli različiti vizualni efekti, omogućujući umjetničku slobodu u kreiranju raznolikih rezultata.

Osim toga, ordered dithering je pristupačan algoritam s relativno jednostavnom implementacijom, što ga čini praktičnim za primjenu u različitim kontekstima. Ova tehnika se može koristiti za poboljšanje vizualne privlačnosti slika ili za prilagodbu prikaza na uređajima s ograničenim kapacitetima boje.

Nažalost, prilikom konverzije može nastati "mrežasti" efekt na određenim slikama, a rezultat ovisi o odabiru veličine threshold matrice. Također, prilikom izračuna vremenske složenosti algoritma, dolazi se do zaključka da je složenost $O(n \cdot m)$, odnosno broj koraka unutar programa ovisi o dimenzijama slike koja se nastoji preobraziti.

U konačnici, ordered dithering predstavlja snažan alat za postizanje umjetničkih i estetskih efekata u obradi slika, nudeći kreativna rješenja za poboljšanje vizualnog dojma i prilagodbu slika prema željenim specifikacijama.

6. Literatura

[1] Wikipedia, *Ordered dithering*, https://en.wikipedia.org/wiki/Ordered_dithering, pristupljeno 25.1.2024.

[2] Youtube kanal Computerphile, *Ordered Dithering*, https://www.youtube.com/watch?v=IviNO7iICTM, pristupljeno 25.1.2024.

[3] Lovrenčić, A., *Prezentacije iz kolegija SPA,* pristupljeno 28.1.2024.