UNIVERSITATEA POLITEHNICĂ DIN BUCUREȘTI

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

DEPARTAMENTUL DE AUTOMATICĂ ȘI INGINERIA SISTEMELOR

PROIECT AWJ

Gray Level Histogram of a Gray-Scale Image

Student:

Iacob Roxana

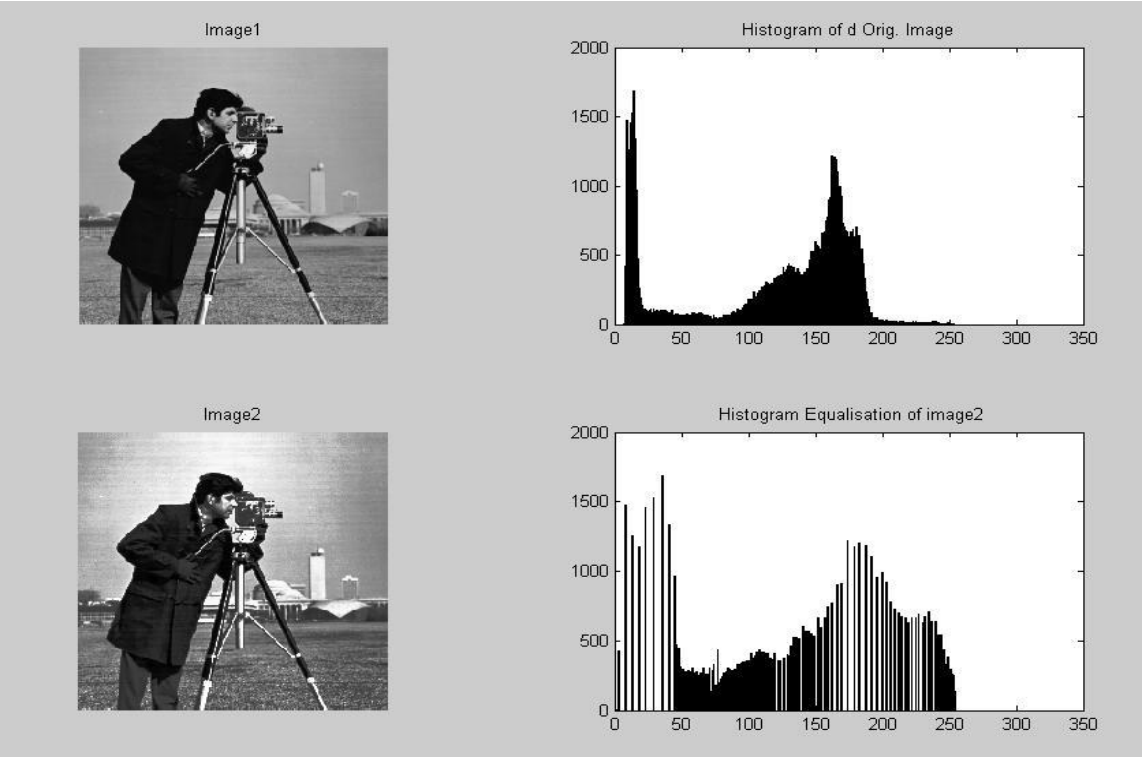
332AB

BUCUREȘTI

2022

Introducere

Proiectul constă în implementarea algoritmului Gray Level Histogram of a Gray-Scale Image (Tema 12). Histograma unei imagini este o diagramă a nivelurilor valorilor de gri față de numărul de pixeli la acea valoare. O histogramă apare ca un grafic, reprezentând „luminozitatea” pe axa orizontală de la 0 la 255 (pentru o scară de intensitate pe 8 biți) și „numărul de pixeli” pe axa verticală.



Sursă imagine: link.imagine (Google) – reprezentarea histogramei cu scop demonstrativ în proiect apare o reprezentare mult mai simplificată

Descrierea funcționalității aplicației

Primul lucru pe care îl face este de a încărca o imagine în format .bmp. După urmează aplicarea algoritmului: o imagine colorată transpusă într-o matrice cu valorile de gri ale pixelilor din imaginea procesata, într-un vector se salvează numărul de apariții a nivelului de gri din matrice, în funcție de valoarea maximă găsită în vectorul de apariții se formează imaginea (tot în format .bmp) cu nivelurile de gri -/+5 valoarea de gri care apare de cele mai multe ori. Într-un fișier sunt scrise numărul de apariții al fiecărui nivel de gri și se încearcă o reprezentare a histogramei folosind semnul (*), reprezentarea e orientativă deoarece afișarea numărului total de * pentru fiecare apariție a fiecărui nivel ar fi fost foarte ineficientă deoarece o valoare poate apărea de foarte multe ori ex, 123400, așa că am afișat un număr mai redus al acestora, împărțind numărul de apariții la un număr destul de mare.

Exemplificare: este doar orientativ

Imagine colorată

Se obține o matrice cu nivelurile de gri

Nivelul de gri al fiecărui pixel se calculează cu formula:

$$Y' = 0.299R' + 0.587G' + 0.114B'$$

204	204	204	255	255
153	204	204	204	255
153	153	153	204	204
102	102	102	153	204
51	0	51	153	204
51	51	51	153	153

Matrice cu nivelurile de gri

Imaginea alb negru

Într-un vector se salvează numărul de apariții al fiecărui nivel de gri

Valorile unui pixel pot fi cuprinse între 0 – 255

Nivel gri	0	1	2	...	21	...	252	253	254	255
Nr. apariții	100	1111	89	...	2706	...	37	3	0	12

Se găsește valoarea maximă: nivelul de gri care apare de cele mai multe ori este: 21 → rezultă ca noua imagine creată va conține doar nivelurile cuprinse între 21 – 5 și 21 + 5 (va conține doar 11 niveluri de gri)

Reprezentarea histogramei

```
Val pixel gri: 18 Nr aparitii: 1573  *
Val pixel gri: 19 Nr aparitii: 2016  *
Val pixel gri: 20 Nr aparitii: 1721  *
Val pixel gri: 21 Nr aparitii: 2706  ***(val max)
Val pixel gri: 22 Nr aparitii: 2150  *
Val pixel gri: 23 Nr aparitii: 1534  *
Val pixel gri: 24 Nr aparitii: 1459  *
Val pixel gri: 25 Nr aparitii: 1509  *
Val pixel gri: 26 Nr aparitii: 1599  *
Val pixel gri: 27 Nr aparitii: 1466  *
Val pixel gri: 28 Nr aparitii: 1563  *
Val pixel gri: 29 Nr aparitii: 1422  *
Val pixel gri: 30 Nr aparitii: 1453  *
Val pixel gri: 31 Nr aparitii: 1510  *
```

Mod de implementare - Descriere

Pentru realizarea proiectului am folosit **2 pachete**, unul în care se găsește **clasa pentru test**, iar în celălalt **restul claselor** ce au fost create pentru implementarea cerințelor, pentru a avea codul mai secționat, fiindu-mi mai ușor să fac modificările corespunzătoare în momentul când aveam nevoie, cât și pentru a fi mai vizibile abordările pentru cerințele propuse de proiect. M-am folosit de 10 clase, acestea sunt:

- Main

Application (aplicația de test): aici au loc operațiile de deschidere/citire a imaginii .bmp, imaginea este procesată utilizând multi-threading (are loc secționarea imaginii în 4 secvențe), se aplică algoritmul pe imaginea procesată și se creează noua imagine. Pentru fiecare proces se **monitorizează timpul de execuție**.

- Clase pentru implementarea algoritmului Gray Level Histogram of a Gray-Scale Image:

Histogram (Algoritumul pentru rezolvarea temei – algoritmul de bază)

ConvertColorToGray (convertește pixelii colorați în nivelul de gri corespunzător - metodă pentru implementarea algoritmului)

GrayLevel (se calculează nivelul e gri pentru fiecare pixel – metodă pentru implementarea algoritmului)

- Clase pentru implementarea cerințelor de proiectare propuse în temă (Moșteniri, Thread-uri, Interfață):

Interface: această clasă este implementată pentru a îndeplini cerința referitoare la crearea unei **interfețe** cât și pentru folosirea unei **metode abstracte**.

Image: această clasă îmi încarcă și îmi creează noua imaginea rezultată din aplicarea algoritmului Histogram (asa se numește clasa). Moștenirea poate fi observată în cadrul acestei clase, **moștenire pe 4 nivele**:

Interface → GrayLevel → ConvertColorToGray → Histogram

```
1  package Package2;
2  // Interface este o interfata ce contine o metoda abstracta
3  public abstract class Interface {
4      public abstract void inheritance(String[] args); // metoda getGrayLevel
5  }
6
```

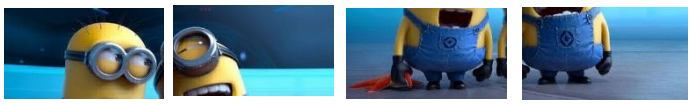
```
public void inheritance(String[] args){
    System.out.println("Mostenire multipla: clasa GrayLevel mosteneste Clasa Interface (1)");
}
```

```
public void inheritance(String[] args){
    System.out.println("Mostenire multipla: clasa ConvertColorToGray mosteneste clasa GrayLevel care mosteneste Clasa Interface (2)");
}
```

```
public void inheritance(String[] args){
    System.out.print("Mostenire multipla: clasa Histogram mosteneste clasa ConvertColorToGray care mosteneste clasa GrayLevel care mosteneste Clasa Interface (3)");
}
```

Multi-threding: pentru secționarea imaginii

ThreadClass, ProducerClass, ConsumerClass, BufferClass



Aceste clase au fost create pe baza laboratoarelor 6 și 7 cât și cu ajutorul cursurilor ce au fost disponibile pe platforma moodle din cadrul cursului de AWJ.

ThreadClass

```
package Package2;
public abstract class ThreadClass extends Thread {
    private boolean consumer;

    public abstract void ThreadStart() throws InterruptedException;

    // constructor
    public ThreadClass(boolean c){
        super();
        consumer = c;
    }

    // afisez cand Thread-ul a inceput sa porneasca si cand s-a term
    // cum am facut la laborator
    public void run(){
        System.out.println("[ThreadClass] Thread-ul a inceput.");
        try{
            this.ThreadStart();
        } catch(InterruptedException e){
            e.printStackTrace();
        }
        System.out.println("[ThreadClass] Thread-ul s-a terminat.");
    }
}
```

ProducerClass

```
package Package2;
public class ProducerClass extends ThreadClass {
    private BufferClass buffer;

    // constructor
    public ProducerClass(BufferClass b){
        super(true);
        buffer = b;
    }

    @Override
    public void ThreadStart(){
        // citesc datele despre imagine si afisez
        // din imagine
        for (int i = 0; i < 4; i++) {
            buffer.readImg();
            System.out.println("Sectiunea" + i);
        }
    }
}
```

ConsumerClass

```
package Package2;
import java.awt.image.BufferedImage;

public class ConsumerClass extends ThreadClass{
    private BufferClass buffer;

    public ConsumerClass(BufferClass b){
        super(false);
        buffer = b;
    }

    @Override
    public void ThreadStart() throws InterruptedException{
        for(int i = 0; i < 4; i++){
            // sectionarea imaginii in 4 secvente
            BufferedImage imgSection = new BufferedImage(buffer.getWidth(), 1
            try {
                imgSection = buffer.writeImg(i);
            } catch (IOException e1) {
                e1.printStackTrace();
            }

            File f = new File("sectiune" + String.valueOf(i + 1) + ".bmp");
            try {
                ImageIO.write(imgSection, "bmp", f); // se scrie fiecare sect
            } catch (IOException e) {
                e.printStackTrace();
            }
            sleep(1000);// laborator
        }
    }
}
```

ThreadClass

```
1  package Package2;
2  import javax.imageio.ImageIO;
7
8  public class BufferClass {
9      private boolean available = false;
10     //argumentele imaginii
11     String imgPath; // calea de acces la imagine
12     private int height = 1, width = 1; // format imagine: inaltime, latime
13     public BufferedImage imageS = null; // pentru citerea fiecarui sfert (1/4)
14
15     public BufferClass(String p){ // constructor
16         imgPath = p;
17     }
18
19     // citirea informatiilor din imaginea sursa (de catre Producer Thread)
20     public synchronized void readImg(){
21
22         File pathF = new File(this.imgPath);
23         try{ // pregatesc imaginea pentru a citi un sfert din ea
24             imageS = ImageIO.read(pathF);
25             // inaltimea si latimea imaginii
26             height = imageS.getHeight();
27             width = imageS.getWidth();
28         } catch (IOException e) {
29             e.printStackTrace();
30         }
31
32         // cum am facut la laborator
33         while (!available) {
34             try {
35                 wait(); // Asteapta Producer sa puna o valoare
36             } catch (InterruptedException e) {
37                 e.printStackTrace();
38             }
39         }
40         available = false;
41         notifyAll();
42     }
43
44     // prelucrarea informatiilor din imaginea sursa primite de la Producer Thr
45     public synchronized BufferedImage writeImg(int section) throws IOException
46         // initializez un vector in care sa retin secventele
47         BufferedImage imgs[] = new BufferedImage[4];
48
49         // cum am facut la laborator
50         while (available) { // fiecare sectiune este scrisa (sunt 4 sectiuni)
51             try {
52                 wait(); // asteapta Consumer sa preaia valoarea
53             } catch (InterruptedException e) {
```

Rezultate

În cele ce va urma voi prezenta practic ce returnează aplicația.

Ex1: Sursă imagine: https:imagine_peisaj

Primul pas: se încarcă imaginea

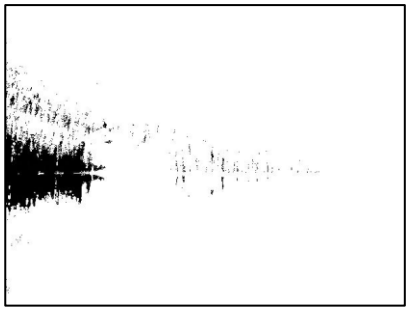
Performanțe

```
Read info about image took 4455 milliseconds
Reading complete.
Image reading stage took 88 milliseconds
[ThreadClass] Thread-ul a inceput.
[ThreadClass] Thread-ul a inceput.
Sectiunea0
Sectiunea1
Sectiunea2
Sectiunea3
[ThreadClass] Thread-ul s-a terminat.
[ThreadClass] Thread-ul s-a terminat.
Multithreading + GrayScaleLevel took 4119 milliseconds
Reading complete.
Writing complete.
Final 163 milliseconds
```

Pas doi: se secționează imaginea

Pas trei: se convertește imaginea colorată în alb negru

Pas patru: se creează imaginea cu nivelurile de gri dorite



Pas cinci: rezultate histogramă

```
File  Edit  Format  View  Help
Val pixel gri: 0 Nr aparitii: 4600  ****
Val pixel gri: 1 Nr aparitii: 3056  ****
Val pixel gri: 2 Nr aparitii: 1319  *
Val pixel gri: 3 Nr aparitii: 948  *
Val pixel gri: 4 Nr aparitii: 831  *
Val pixel gri: 5 Nr aparitii: 696  *
Val pixel gri: 6 Nr aparitii: 695  *
Val pixel gri: 7 Nr aparitii: 667  *
Val pixel gri: 8 Nr aparitii: 669  *
Val pixel gri: 9 Nr aparitii: 692  *
Val pixel gri: 10 Nr aparitii: 734  *
Val pixel gri: 11 Nr aparitii: 757  *
Val pixel gri: 12 Nr aparitii: 788  *
Val pixel gri: 13 Nr aparitii: 832  *
Val pixel gri: 14 Nr aparitii: 864  *
Val pixel gri: 15 Nr aparitii: 858  *
Val pixel gri: 16 Nr aparitii: 918  *
Val pixel gri: 17 Nr aparitii: 920  *
Val pixel gri: 18 Nr aparitii: 1038  *
Val pixel gri: 19 Nr aparitii: 1257  *
Val pixel gri: 20 Nr aparitii: 1324  *
Val pixel gri: 21 Nr aparitii: 1326  *
```

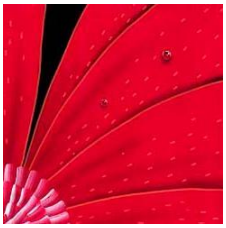
Ex2:

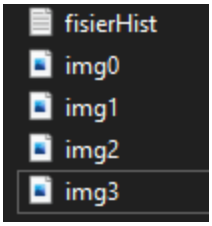
Se încarcă imaginea:

Performanțe

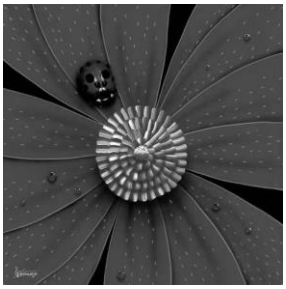
```
Read info about image took 1945 milliseconds
Reading complete.
Image reading stage took 1718 milliseconds
[ThreadClass] Thread-ul a inceput.
[ThreadClass] Thread-ul a inceput.
Sectiunea0
Sectiunea1
Sectiunea2
Sectiunea3
[ThreadClass] Thread-ul s-a terminat.
[ThreadClass] Thread-ul s-a terminat.
Multithreading + GrayScaleLevel took 4226 milliseconds
Reading complete.
Writing complete.
Final 189 milliseconds
```

Se secționează imaginea

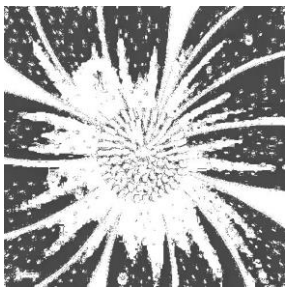





Se convertește imaginea colorată în alb negru



Se creează imaginea cu nivelurile de gri dorite



Rezultate histogramă

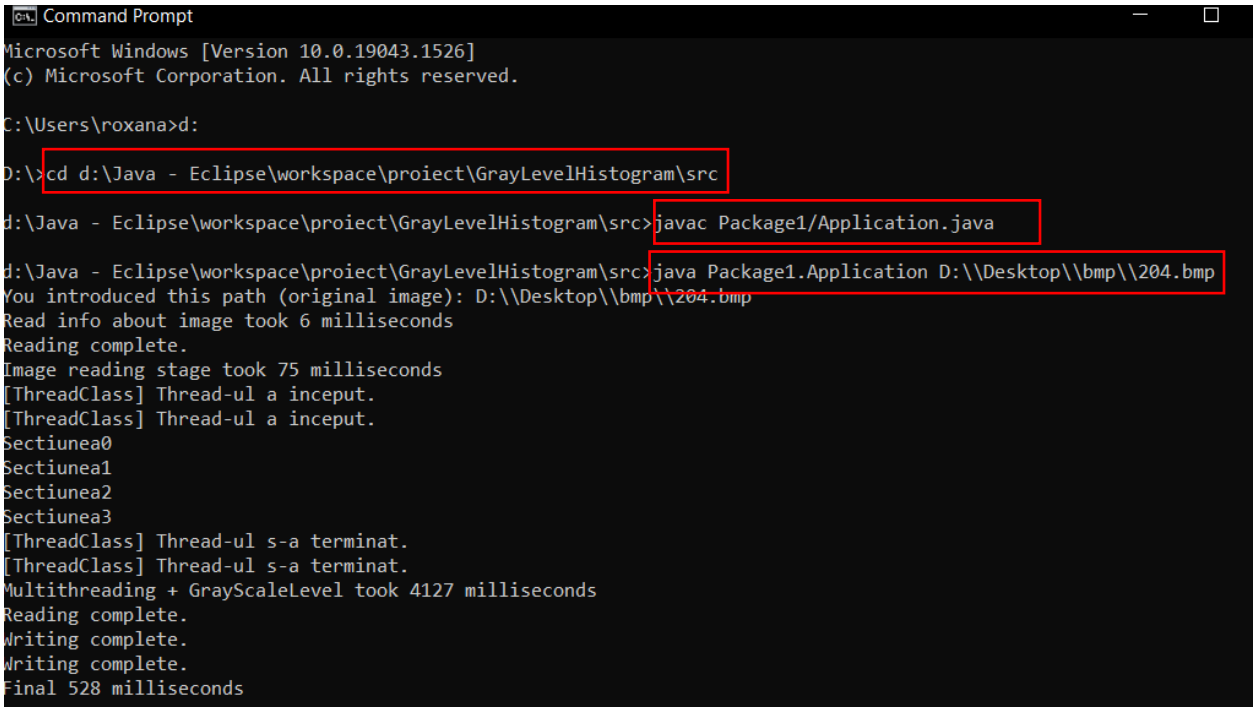
```
Val pixel gri: 46 Nr aparitii: 685  *
Val pixel gri: 47 Nr aparitii: 807  *
Val pixel gri: 48 Nr aparitii: 835  *
Val pixel gri: 49 Nr aparitii: 1006  *
Val pixel gri: 50 Nr aparitii: 1073  *
Val pixel gri: 51 Nr aparitii: 1141  *
Val pixel gri: 52 Nr aparitii: 1260  *
Val pixel gri: 53 Nr aparitii: 1508  *
Val pixel gri: 54 Nr aparitii: 1555  *
Val pixel gri: 55 Nr aparitii: 1842  *
Val pixel gri: 56 Nr aparitii: 2207  *
Val pixel gri: 57 Nr aparitii: 2431  *
Val pixel gri: 58 Nr aparitii: 3006  ****
Val pixel gri: 59 Nr aparitii: 3267  ****
Val pixel gri: 60 Nr aparitii: 3610  ****
Val pixel gri: 61 Nr aparitii: 4029  ****
Val pixel gri: 62 Nr aparitii: 4090  ****
Val pixel gri: 63 Nr aparitii: 4796  ****
Val pixel gri: 64 Nr aparitii: 5440  ****
Val pixel gri: 65 Nr aparitii: 5830  ****
Val pixel gri: 66 Nr aparitii: 6886  *****
Val pixel gri: 67 Nr aparitii: 7052  *****
Val pixel gri: 68 Nr aparitii: 8338  *****
Val pixel gri: 69 Nr aparitii: 8527  *****
Val pixel gri: 70 Nr aparitii: 10824  ******
Val pixel gri: 71 Nr aparitii: 12427  *******
Val pixel gri: 72 Nr aparitii: 10724  ******
Val pixel gri: 73 Nr aparitii: 20578  *********
Val pixel gri: 74 Nr aparitii: 18275  *********
Val pixel gri: 75 Nr aparitii: 14778  *******
Val pixel gri: 76 Nr aparitii: 9453  ******
Val pixel gri: 77 Nr aparitii: 4351  ****
Val pixel gri: 78 Nr aparitii: 2244  *
Val pixel gri: 79 Nr aparitii: 1417  *
Val pixel gri: 80 Nr aparitii: 1112  *
Val pixel gri: 81 Nr aparitii: 896  *
```

Testarea Funcționalității (cum se realizează)

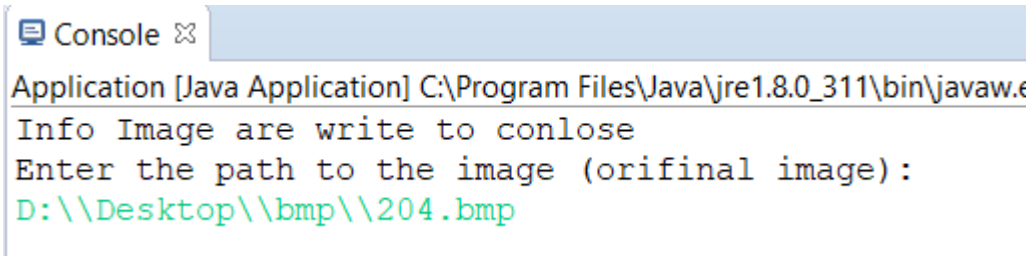
Testarea se poate face atât prin argumente în linia de comandă, cât și din consolă.

Utilizatorul introduce în linia de comandă / consolă calea către imaginea ce se dorește a fi procesată.

Ex: linie de comandă: aplicația se rulează din directorul unde se află



Ex: consolă: se deschide aplicația → run → introducere cale



Concluzii Finale

Rezolvarea temei se putea face mult mai ușor, utilizând doar algoritmul din clasa Histogram și celelalte clase care conțin funcții esențiale aplicării algoritmului (ConvertColorToGray, GrayLevel). Prin metoda aceasta de rezolvare, folosind moșteniri și thread-uri, am învățat mult mai bine aceste concepte din Java, fiind nevoită să le aplic am înțeles mai bine cum se utilizează și cum funcționează. A fost un proiect interesant, care pe lângă cunoștințele tehnice legate de implementarea lui, ne-a deschis noi orizonturi în ceea ce înseamnă procesarea de imagini.

Bibliografie

Documentația folosită pentru realizarea proiectului constă atât în cursurile de pe moodle de la cursul AWJ, cât și alte surse găsite pe Internet, acestea fiind:

https://www.cis.rit.edu/people/faculty/pelz/courses/SIMG203/res.pdf

https://stackoverflow.com/questions/2615522/java-bufferedimage-getting-red-green-and-blue-individually

https://studylib.net/doc/7883468/grey_level_enhancement

https://uomustansiriyah.edu.iq/media/lectures/9/9_2017_09_27!04_29_16_PM.pdf

https://www.stemmer-imaging.com/en/knowledge-base/grey-level-grey-value/