Universitatea POLITEHNICA din București

Procesare paralelă de imagini

FACULTATEA DE AUTOMATICĂ & CALCULATOARE

Universitatea POLITEHNICA din București

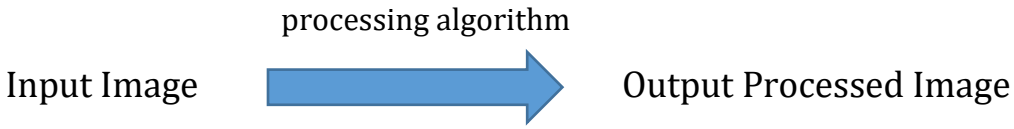
Cuprins:

FACULTATEA DE AUTOMATICĂ & CALCULATOARE
Universitatea POLITEHNICA din București

1. Introducere, prezentare a temei, prezentare a obiectivelor

Acest proiect are ca scop prezentarea unei teme de procesare a unor imagini și conține o parte teoretică care va descrie procedeele de „histogram stretching", conversie a unei imagini la imagine grayscale, conversie la imagine negativă, conversie la imagine sepia, oglindire a imaginii și rotire a acesteia cu un anumit unghi, dar și o parte practică, reprezentată de o aplicație implementată în Java care realizează efectiv operațiile date. În cadrul acestuia, vor fi prezentate concepte bine-cunoscute din teoria procesării imaginilor, precum histograma unei imagini, intensitatea unui pixel, contrastul unei imagini și modelul de culoare RGB. Descrierea acestora este imperios necesară pentru înțelegerea modului de implementare a aplicației.

Aplicația este, așadar, implementată în Java și are ca scop realizarea operațiilor ,menționate mai sus, pentru o imagine dată. Ea este multimodulară, având o funcționalitate precisă – primește ca input o imagine de tip BMP, aplică operația de corespunzătoare imaginii și va avea ca output (rezultatul rulării aplicației) o nouă imagine de tip BMP procesată. La finalul execuției aplicației, în directorul în care se află imaginea originală neprocesată, vor mai exista încă 6 imagini procesate care atestă funcționalitatea corectă a acesteia.



Aplicația are la bază paradigma Producer-Consumer (cazul un producător și un consumator), deci este multithreading. Mai exact, în plus față de firul de execuție principal, în cadrul programului mai rulează alte 8 thread-uri – 1 producer, 1 consumer și 6 thread-uri de procesare efectivă a imaginii. Mai multe detalii legate de implementarea acestui mecanism de multithreading vor fi expuse în capitolul de implementare a aplicației.
Etapele de execuție ale aplicației sunt:

- se citesc informațiile de identificare a fișierului sursă, un nivel de procesare al imaginii dorit (va fi pomenit ca nivel de stretching în cadrul acestei documentații) și un grad al unghiului de rotire a imaginii
- se citește fișierul sursă

- se procesează imaginea
- se scrie fișierul destinație

La fiecare etapă, se va înregistra timpul de execuție și se vor afișa mesaje corespunzătoare la consolă în vederea aprecierii performanțelor acesteia. De asemenea, aplicația precizează la fiecare pas că imaginea a fost citită, procesată și scrisă pe disc prin mesaje afișate tot la consolă.
Aplicația respectă toate principiile de programare orientată pe obiecte – încapsulare, moștenire, polimorfism și SRP (Single Responsability Principle). Fiecare clasă sau interfață implementată în cadrul aplicației are un singur scop precis, așadar aplicația are o funcționalitate și o structură simplă.

2. Prezentarea suportului tehnic, trecere în revistă a unor realizări similare

Pentru a înțelege fiecare operație de procesare menționată mai sus, este necesară prezentarea următoarelor concepte generale din teoria procesării de imagini: contrastul unei imagini și histograma unei imagini.

Așa cum este prezentat în [6], **contrastul** reprezintă diferența dintre intensitatea maximă și cea minimă a pixelilor.

Conform aceleași surse, **histograma unei imagini** este o reprezentare grafică a valorilor nivelurilor de gri în raport cu numărul de pixeli cu aceste valori.

O histogramă apare ca un grafic cu "luminozitate" pe axa orizontală cu valori de la 0 la 255 (pentru o scală de intensitate pe 8 biți) și "numărul de pixeli " pe axa verticală. Pentru fiecare imagine colorată se calculează trei histograme, câte una pentru fiecare componentă (RGB sau HSL). Histograma ne oferă o reprezentare convenabilă - ușor de citit - a concentrației pixelilor versus luminozitatea unei imagini. Folosind acest grafic putem observa următoarele aspecte:

- dacă o imagine este întunecată sau luminoasă sau dacă are contrast înalt sau scăzut
- ce modificare a contrastului trebuie aplicată în mod corespunzător pentru a face imaginea mai ușor de interpretat prin operațiunile de analiză de către un observator

Ca exemplu, considerăm o imagine, îi construim histograma și îi calculăm contrastul.

Imaginea căreia îi calculăm contrastul și reprezentăm histograma este următoarea:

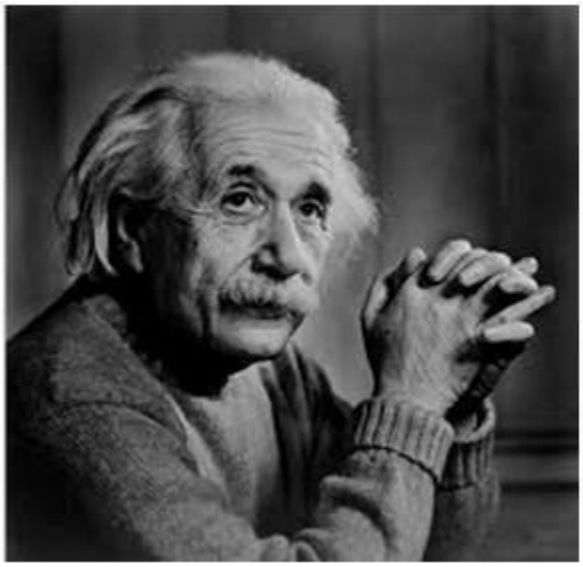


Figura 1: Imagine de input pentru reprezentarea histogramei

Histograma imaginii este:

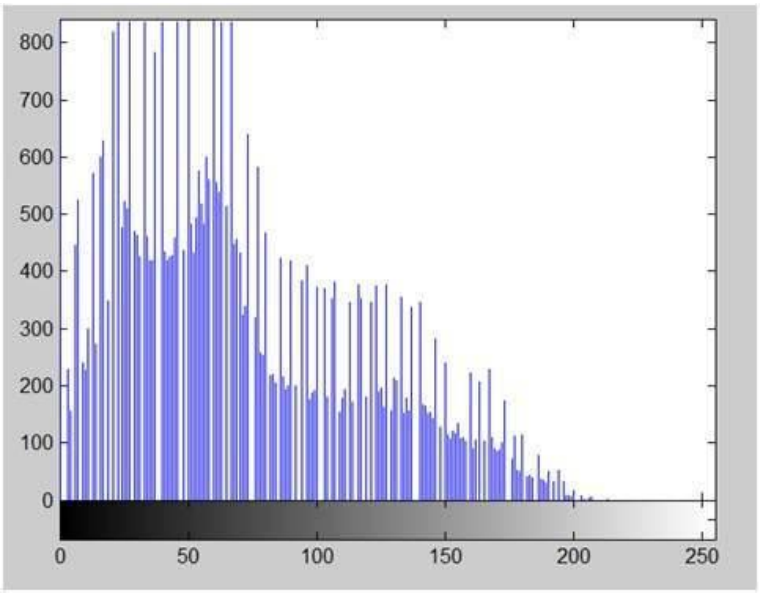


Figura 2: Histograma imaginii anterioare

Contrastul imaginii considerate este 225 (intensitatea maximă a pixelilor este aproximativ 225 și intensitatea minimă a pixelilor este aproximativ 0).

Așa cum este prezentat în [3], în următoarea imagine putem observa aceeași imagine cu 3 contraste diferite. Diferența de contrast se poate observa în histogramele acestora.

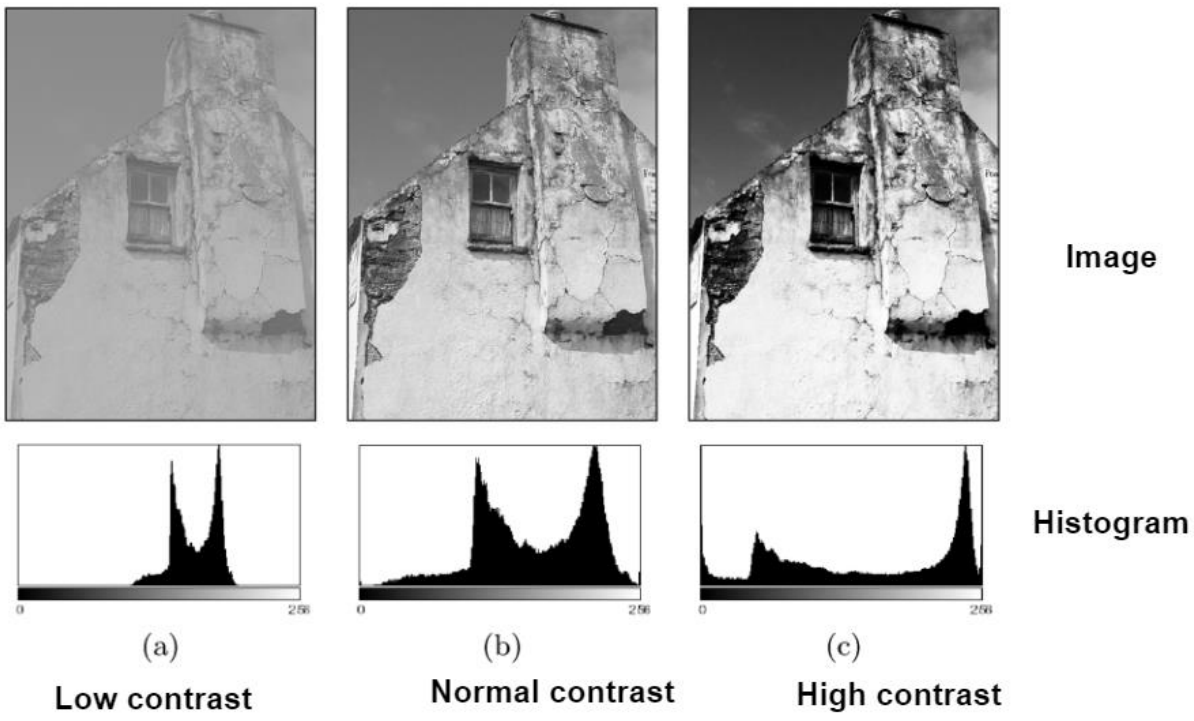


Figura 3: Diferența de contrast explicată la nivel de histograme ale imaginilor

Pentru o imagine se pot crea, de asemenea, și 3 histograme diferite pentru fiecare din cele 3 canale RGB (roșu, verde și albastru):

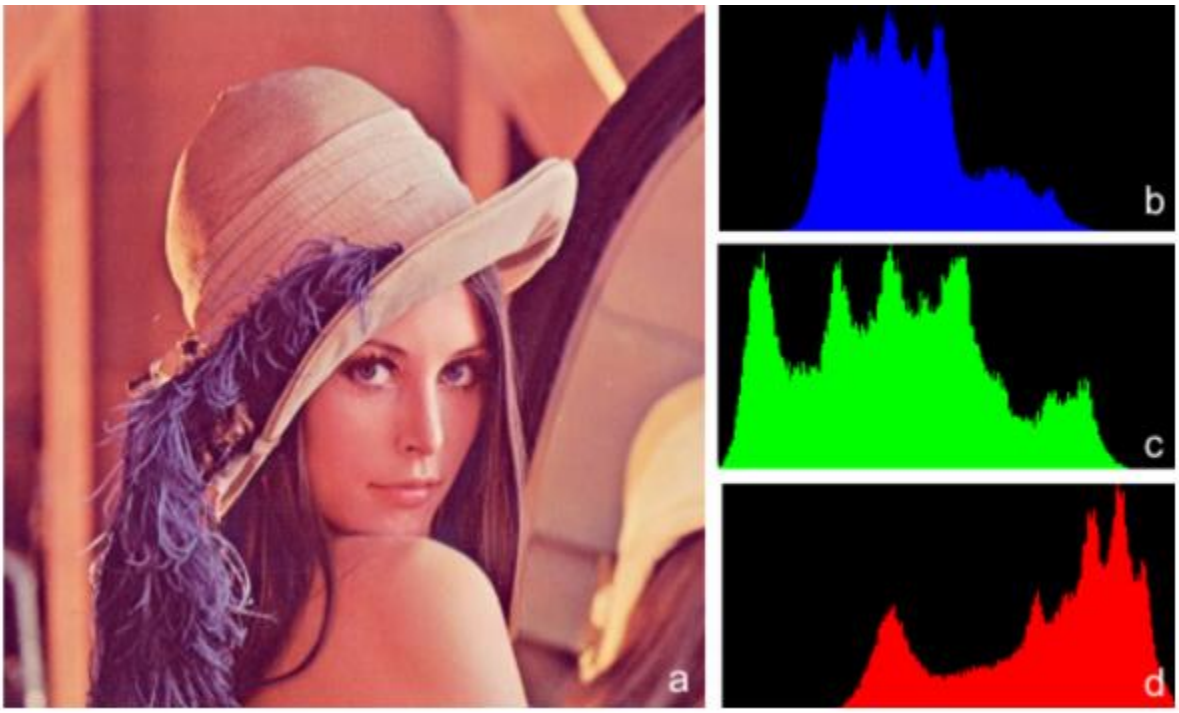


Figura 4: Histograma RGB pe cele 3 canale a celebrei imagini Lena

Mai puțin cunoscută este histograma HSV (Hue, Saturation, Value), care nu va folosită în cadrul implementării aplicației din cadrul acestui proiect.

Histograma RGB

În acest proiect folosim histograma RGB, care este mult mai des întâlnită. Conform [2], imaginile pot fi reprezentate în spațiul RGB al culorilor, în care fiecare pixel este definit de un triplet (r, g, b) care reprezintă intensitatea roșului, a verdelui și, respectiv, a albastrului. Un mod prin care poate fi reprezentată distribuția culorilor într-o imagine este prin analizarea unei histograme, precum cea din figura anterioară. O histogramă măsoară frecvența de apariție a pixelilor care au anumite valori pentru r, g sau b. Se observă că există câte o histogramă separată pentru fiecare dimensiune a spațiului culorilor (respectiv pentru roșu, verde și albastru). Un model folosit frecvent este cel în care valorile lui r, g, b sunt întregi din intervalul [0, 255]. O histograma pentru R, de exemplu, poate avea 256 de valori pe axa orizontală, caz în care vom obține pe a i-a verticală numărul de pixeli din componența R care au valoarea i.

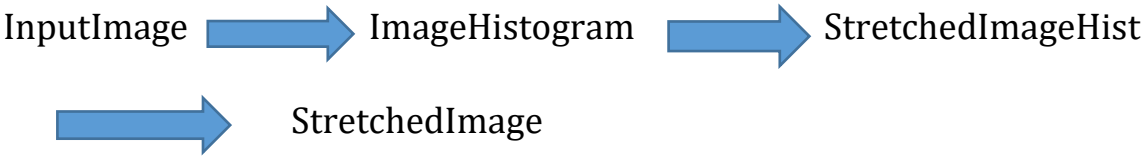
Dacă pe axa orizontală există **count_bins** valori, atunci pe a i-a verticală se va regăsi numărul de pixeli care au valoarea componentei R în intervalul:

$$i \cdot \frac{256}{count\_bins}, \quad i \cdot \frac{256}{count\_bins} + \frac{256}{count\_bins}$$

În mod analog se procedează și pentru componentele G și B ale imaginii.

Histogram Stretching

Prin definiție, așa cum apare în [1] și [6], „histogram stretching" reprezintă o tehnică de creștere a contrastului unei imagini, creștere care se calculează după o formulă specifică asociată fiecărui pixel, modificările realizându-se prin intermediul histogramei imaginii.



În general, funcția de „histogram stretching" este dată prin următoarea relație:

$$\text{Stretch (I (r, c))} = \left[\frac{I(r,c) - I(r,c)_{\min}}{I(r,c)_{\max} - I(r,c)_{\min}} \right] [MAX\text{-}MIN] + MIN.$$

unde:

- $I(r,c)_{max}$ este cel mai mare nivel de gri din imaginea $I(r,c)$
- $I(r,c)_{min}$ este cel mai mic nivel de gri din imaginea $I(r,c)$
- MAX și MIN corespund valorilor maxime și minime de gri – nivel posibil (pentru o imagine pe 8 biți acestea sunt 255 și 0)

Această relație se poate aplica pe o imagine și va realiza stretch-ul histogramei pe întreaga gamă de nivel de gri și are ca efect creșterea contrastului imaginii.

Mai simplu, dacă **f(x,y)** este valoarea intensității fiecărui pixel al imaginii și **g(x,y)** este valoarea intensității fiecărui pixel al imaginii rezultate în urma operației de „histogram stretching", atunci:

$$g(x,y) = \frac{f(x,y)-fmin}{fmax-fmin} * 2^{bpp}$$

După cum se poate observa, formula necesită găsirea intensității minime și maxime a pixelilor. Fracția din formulă este înmulțită cu numărul de niveluri de gri, iar în cazul în care imaginea este 8bpp, spre exemplu, se va înmulți cu 256.
Pentru imaginea cu Einstein de mai sus, formula ia următoarea formă:

$$g(x,y) = \frac{f(x,y)-0}{225-0} * 255$$

Din formulă se observă clar faptul că operația eșuează atunci când $fmax = fmin$ și când există pixeli cu intensitățile 0 și 255 în imagine. Din această cauză, în cadrul algoritmului care va fi inclus în cadrul implementării aplicației, se vor exclude aceste cazuri, impunându-se o restricție în acest sens (nu are sens să includem aceste cazuri întrucât imaginii nu i se va aplica operația de stretching).
Vom prezenta câteva rezultate în care operația de „histogram stretching” are succes:

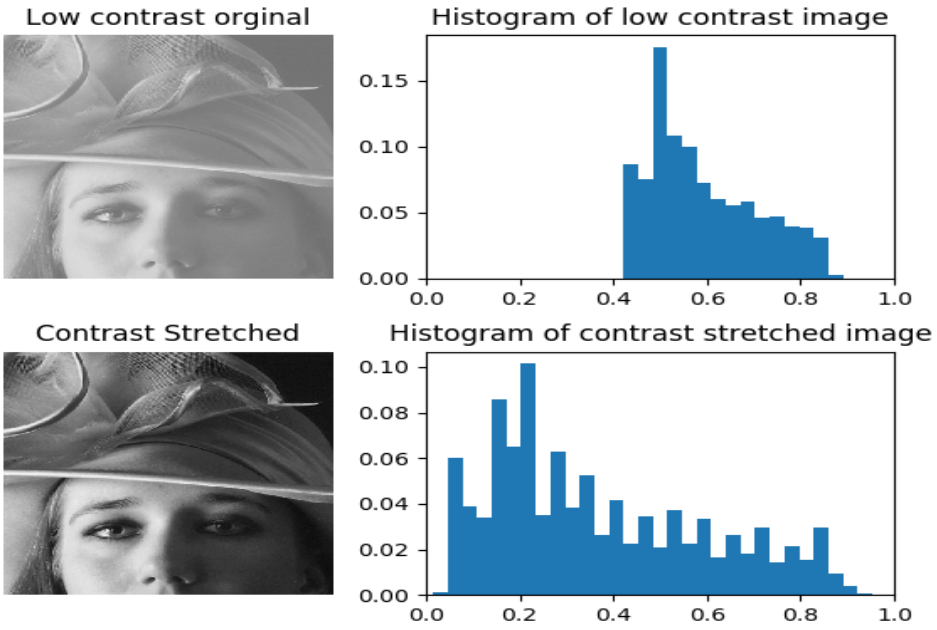


Figura 5: Imaginea originală, imaginea „contrast stretched” și histogramele lor

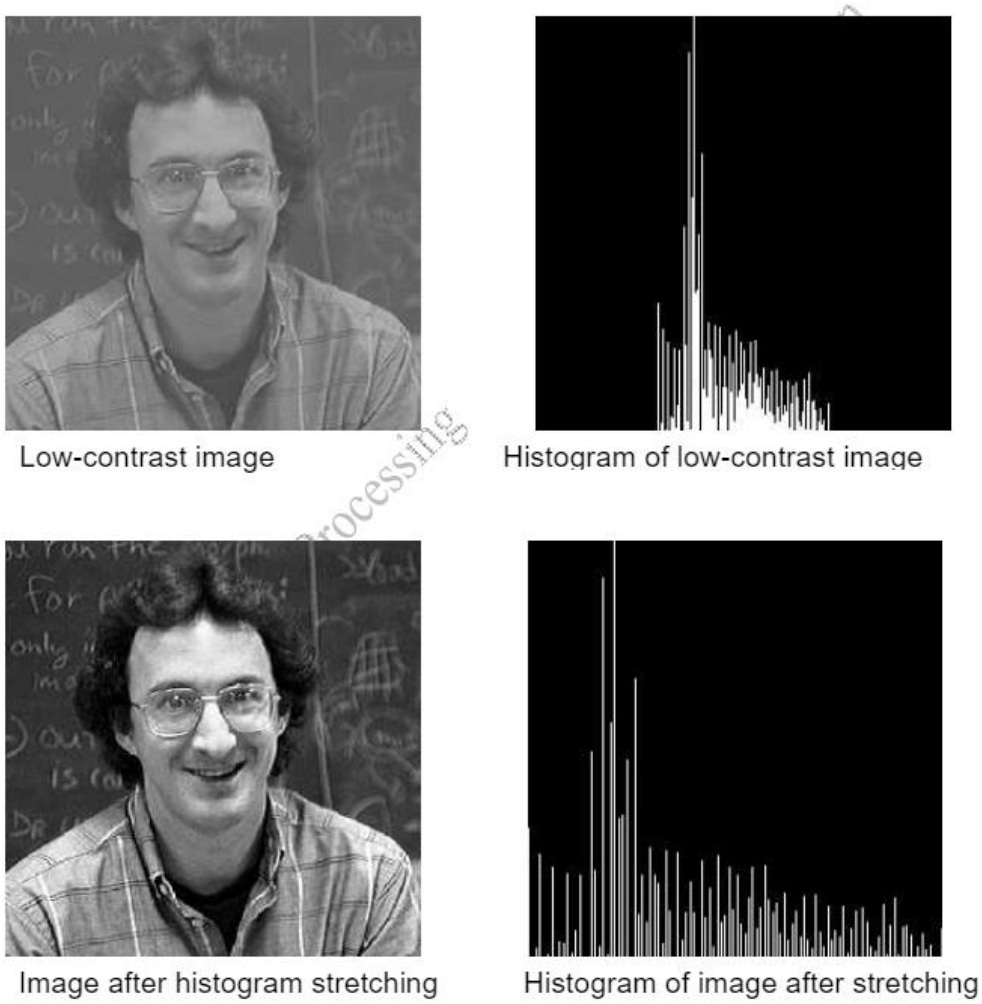


Figura 6: alt exemplu de „histogram stretching” realizat cu succes

De asemenea, prezentăm 2 dintre perechile (inputImage, outputStretchedImage) prelucrate de aplicația ce urmează a fi prezentată mai în detaliu. Pentru prelucrarea acestora, aplicația este adaptată, întrucât formula de stretching folosește și un anumit coeficient de realizare a operației (**discardRatio**), denumit *nivel de stretching* în acest caz.
Pentru un nivel de stretching egal cu 0.1 (10%), avem următoarele rezultate pentru 2 imagini color:

Figura 7: Input image 1

Figura 8: Output 10% stretched image 1

Figura 9: Input image 2

Figura 10: Output 10% stretched image 2

Conversie la imagine grayscale

Așa cum apare în [1] și [2], o imagine în tonuri de gri (sau grayscale) este, pur și simplu, o imagine în care singurele culori sunt nuanțe de gri. Motivul pentru diferențierea acestor imagini de orice alt tip de imagine color este că trebuie furnizate mai puține informații pentru fiecare pixel. De fapt, o nuanță "gri" este una în care componentele roșu, verde și albastru au o intensitate egală în spațiul RGB și, prin urmare, este necesar să se specifice o singură valoare de intensitate pentru fiecare pixel, spre deosebire de cele trei intensități necesare pentru a specifica fiecare pixel într-o imagine color completă.

Adesea, intensitatea tonurilor de gri este stocată ca un întreg pe 8 biți, oferind 256 (2^8) de nuanțe posibile diferite de gri de la negru la alb. Dacă nivelurile sunt distanțate uniform, atunci diferența dintre nivelurile de gri succesive este semnificativ mai bună de folosit în practică decât puterea ochiului uman de detectare corectă a nivelului de gri.

Imaginile în tonuri de gri sunt foarte frecvente pentru că o mare parte din hardware-ul de afișare și captare a imaginilor de astăzi poate suporta doar imagini pe 8 biți. În plus, imaginile în tonuri de gri sunt în întregime suficiente pentru multe sarcini și, de aceea, nu este nevoie să utilizați imagini color mai complicate și mai greu de procesat.

Algoritmul de conversie a imaginii originale color la imagine grayscale este destul de simplu și conține următorii pași descriși și în [5]:

- Se obține întâi valoarea RGB a pixelului
- Se calculează media RGB cu ajutorul unei formule, medie = (R + G + B) / 3
- Se înlocuiesc valorile R, G și B ale pixelului cu media calculată la pasul anterior
- Se repetă pasul 1 la pasul 3 pentru fiecare pixel al imaginii

Un exemplu de conversie la imagine grayscale:

Figura 11: Input image 1

Figura 12: Output grayscale image 1

Conversie la imagine negativă

Conform [4], negativul unei imagini se realizează prin înlocuirea intensității **I** din imaginea originală cu **I-1**, adică cei mai întunecați pixeli vor deveni cei mai luminoși, iar cei mai luminoși pixeli vor deveni cei mai întunecați. Imaginea negativă este produsă scăzând fiecare pixel din valoarea intensității maxime.

De exemplu, într-o imagine în tonuri de gri pe 8 biți, valoarea intensității maxime este de 255, astfel încât fiecare pixel este scăzut din 255 pentru a produce imaginea de ieșire.

Algoritmul de conversie la imagine negativă este următorul:

- Se obține întâi valoarea RGB a pixelului utilizând metoda
- Se calculează noile valori RGB după urmează:

  R’ = 255 – R
  G’ = 255 – G
  B’ = 255 – B
- Se înlocuiesc valorile R, G și B ale pixelului cu valorile R’, G’ și B’ calculate în pasul anterior
- Se repetă pașii de la 1 la 3 pentru fiecare pixel al imaginii

Un exemplu de conversie la imagine negativă:

Figura 13: Input image 1

Figura 14: Output negative image 1

Conversie la imagine Sepia

Așa cum apare în [3], filtrul Sepia este unul dintre cele mai populare instrumente pentru editarea imaginilor. Cuvântul "sepia" se referă la numele pigmentului maro care a fost utilizat pe scară largă de fotografi în primele zile ale fotografiei. Efectul Sepia conferă imaginilor un ton maroniu cald. Filtrul Sepia îmbunătățește aspectul general al imaginii originale.

Totodată conform [2], termenul Sepia, atunci când este utilizat în contextul fotografiei, se referă la o imagine monocromă redată în tonuri de maro, mai degrabă decât la tonurile de gri utilizate într-o imagine tradițională alb-negru. Imaginile Sepia au fost produse inițial prin adăugarea unui pigment la o imprimare pozitivă în timp ce expuneau o imagine capturată pe film. Cuvântul este inspirat de la peștele sepie, a cărui cerneală a fost folosită pentru a crea colorant maro folosit pentru a modifica tonul de imagini Sepia. În fotografia digitală, o imagine poate fi transformată într-o imagine Sepia folosind software-ul digital de procesare a imaginilor.

Algoritmul de conversie la imagine Sepia este:

- Se obține valoarea RGB a pixelului
- Se calculează noile valori ale R, G și B ale pixelului.
- Se setează noile valori ale valorilor R, G și B a pixelului în conformitate cu următoarele condiții:

  Dacă valoarea newRed > 255, setați R = 255, altfel set R = newRed
  Dacă noua valoareGreen > 255, setați G = 255, altfel set G = newGreen
  Dacă valoarea newBlue > 255, setați B = 255, altfel set B = newBlue
- Se înlocuiesc valorile R, G și B cu noile valori calculate pentru pixel în pasul anterior
- Se repetă pașii de la 1 la 4 pentru fiecare pixel al imaginii

Un exemplu de conversie la imagine Sepia:

Figura 15: Input image 1

Figura 16: Output Sepia image 1

Image mirroring

Conform [1], o imagine în oglindă este o duplicare reflectată a unui obiect care pare aproape identic, dar este inversată în direcția perpendiculară pe suprafața oglinzii. Acest efect optic apare, spre exemplu, în cazul în care subiectul privește în oglindă sau în apă.

Algoritmul de conversie la imagine oglindită este:

- Se obțin valorile ARGB (Alfa, Roșu, Verde și Albastru) din imaginea de intrare în direcția de la stânga la dreapta
- Se setează valorile ARGB (Alfa, Roșu, Verde și Albastru) pentru o imagine cu aceleași dimensiuni ca imaginea originală, creată în prealabil, în direcția de la dreapta la stânga
- Se repetă pașii anteriori pentru fiecare pixel al imaginii

Un exemplu de image mirroring:

Figura 17: Input image 1

Figura 18: Output mirrored image 1

Rotirea unei imagini cu un anumit unghi

Așa cum apare în [6], rotația imaginii este o rutină comună de procesare a imaginilor. Operația de rotație a imaginii are nevoie ca intrări de imaginea originală, unghiul de rotație θ și un punct în jurul căreia se face rotația. În acest caz, vom considera acest punct originea planului (punctul de coordonate 0 și 0).

Algoritmul de rotire a unei imagini este:

- Se obține valoarea RGB a pixelului
- Se află noua locație din matrice a pixelului folosind ca referință unghiul de rotire primit ca dată de intrare. Destinația se află folosind o formulă simplă din geometrie

$$\begin{bmatrix} x^* \\ y^* \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

- Se setează noua valoare a pixelului, calculată anterior
- Se repetă pașii 2 și 3 pentru fiecare pixel al imaginii

Un exemplu de rotire cu unghi de 180 de grade:

Figura 19: Input image 1

Figura 20: Output rotated image 1

3. Prezentare tehnică a etapei de implementare

Descriere structurală (a modulelor)

Aplicația este implementată pe baza a 13 clase și o intefață. Cele 13 clase sunt următoarele:

1. **Main** – clasa principală utilizată pentru citirea și scriere din și în fișiere, dar și pentru procesare (în ea sunt create thread-urile, unul fiind de procesare efectivă); ea conține metoda main în care sunt realizate operațiile I/O și creare de obiecte necesare implementării (semafoare, imaginile etc.)
2. **ThreadClass** – clasa principală de thread, clasă abstractă, părinte ale celor 3 thread-uri specifice
3. **HistogramBuffer** – clasa buffer care stochează histogramele imaginilor (se implementează un buffer de capacitate 1)
4. **Producer** – thread-ul producător care produce/pune sferturi din histograma imaginii în buffer
5. **Consumer** – thread-ul consumator care consumă/preia sferturi din histograma imaginii din buffer și reconstruiește histograma imaginii originale

6. **StretchingThread** – thread-ul care realizează operația de „histogram stretching” asupra imaginii prin intermediul histogramei preluate de Consumer
7. **ImageHistogram** – clasa care construiește histograma imaginii prin intermediul metodei getImageHistogram
8. **StretchingTimeTable** – clasă copil a lui StretchingThread și e folosită mai mult în scop didactic pentru a afișa timpii de procesare pentru fiecare etapă în parte prin intermediul metodei varargs getStretchingStepsTime
9. **ConvertingToGrayscaleThread** - thread-ul care realizează operația de conversie la imagine grayscale
10. **ConvertingToNegativeThread** - thread-ul care realizează operația de conversie la imagine negativă
11. **ConvertingToSepiaThread** - thread-ul care realizează operația de conversie la imagine Sepia
12. **ImageMirroringThread** - thread-ul care realizează operația de image mirroring asupra imaginii originale
13. **RotatingImageThread** - thread-ul care realizează operația de rotire cu un anumit unghi asupra imaginii originale

Interfața **Image** conține o metodă care notifică crearea histogramei pentru o imagine și este implementată de clasa ImageHistogram.

Cerințe de implementare ale aplicației:

1. Aplicația folosește pentru input și output imagini de tip 24bit BMP – RGB.
2. Pentru operația de „histogram stretching” e folosită formula din secțiunea ***2. Prezentarea suportului tehnic, trecere în revistă a unor realizări similare***, dar și algoritmi low-level bazați pe calculul noilor valori ale pixelilor (în clasa StretchingThread) și pe construirea histogramei RGB a unei imagini (în clasa HistogramImage). De asemenea, celelalte 5 operații sunt realizate pe baza algoritmilor menționați în secțiunea trecută.
3. Toate conceptele OOP sunt consistent integrate în implementarea aplicației. Există **încapsulare** la nivelul claselor care conțin diverse metode și obiecte necesare procesării (de exemplu, clasa Producer conține un obiect buffer de tip HistogramBuffer și alte metode specifice), **moștenire** – există o **ierarhie de clase** (clasele Producer, Consumer, StretchingThread, ConvertingToGrayscaleThread, ConvertingToNegativeThread, ConvertingToSepiaThread, ImageMirroringThread și RotatingImageThread moștenesc clasa ThreadClass, care la rândul ei moștenește clasa Thread) și **3 niveluri de moștenire** – clasa StretchingTimeTable moștenește clasa StretchingThread, care moștenește clasa ThreadClass, care la rândul ei moștenește

clasa Thread), **polimorfism** – există *metode getters* implementate în cadrul claselor (de exemplu, clasele de thread moștenesc metoda getThreadName din clasa principală ThreadClass) și **abstractizare** – există **clasa abstractă** ThreadClass care conține **metoda abstractă** getThreadName.

4. Codul sursă este în întregime comentat și coding style-ul este adaptat limbajului Java.
5. Aplicația folosește operațiile de I/O pe fișiere, la citirea și la scrierea imaginii (se poate observa în cadrul clasei principale Main).
6. Există **operații de intrare de la tastatură** – este citit nivelul de stretching și gradul de rotire a imaginii și **prin parametrii liniei de comandă** – este furnizat astfel numele fișierului de intrare
7. Aplicația este multimodulară, în sensul că există o **ierarhie de clase și cel puțin 3 niveluri de moștenire**.

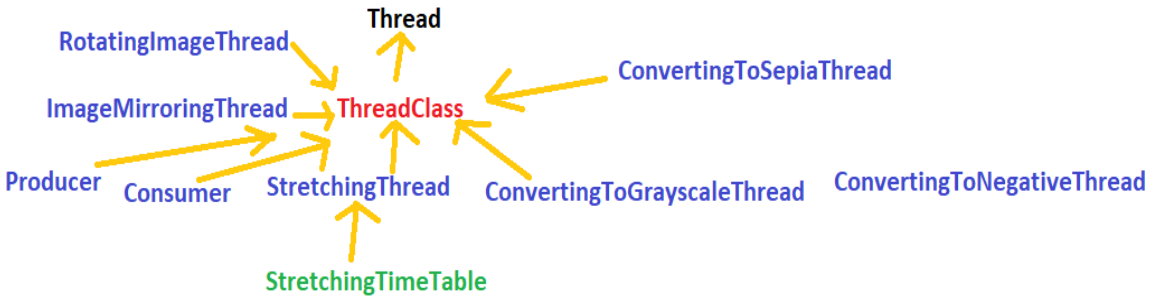


Figura 21: Ierarhia de clase pe 3 niveluri care există în implementarea aplicației

8. Există **metoda varargs** getStretchingStepsTime din clasa StretchingTimeTable.
9. Există **inferfața** Image implementată de clasa ImageHistogram.
10. Există **clasa abstractă** ThreadClass, clasă părinte a claselor concrete Producer, Consumer și StretchingThread și celelalte 5 clase de thread-uri de procesare. Ea conține **metoda abstractă** getThreadName.
11. Aplicația folosește în clase blocuri try-catch pentru toate cazurile de exception/error handling.
12. Aplicația este multithreading cu următoarele funcționalități:
    - thread-ul Producer creează histograma RGB a imaginii citite, pune câte ¼ din histogramă în buffer (obiectul de tip HistogramBuffer) și intră în sleep după fiecare adăugare; acesta nu poate să mai pună în buffer dacă e plin, adică dacă conține un sfert de histogramă deja care nu a fost preluat de Producer și așteaptă până se golește

- thread-ul Consumer preia din buffer câte ¼ din histogramă cât timp acesta nu e gol și intră în sleep după fiecare preluare; dacă buffer-ul este gol, așteaptă până când Producer a mai pus un sfert de histogramă; el reconstruiește histograma pentru procesare la sfârșitul preluării tuturor celor 4 sferturi

- Producer și Consumer comunică printr-un mecanism de wait-notify și există o sincronizare realizată cu synchronized, delimitând astfel zona critică în care se poate afla doar unul dintre aceștia la un anumit moment de timp

- StretchingThread realizează operația efectivă de stretching; pentru a asigura faptul că acesta are ce procesa, adică Consumer a preluat toate cele 4 sferturi ale histogramei de la Producer, am folosit un semafor care impune ca StretchingThread să aștepte execuția Producer-Consumer

- Am folosit sleep pentru a evidenția etapele comunicării

- În toate thread-urile sunt înregistrați timpii de execuție; în cel de procesare sunt înregistrați timpii de execuție pentru ambele etape de procesare: prima etapă – aflarea intensităților minime și maxime ale pixelilor imaginii pe fiecare canal R, G, B și a doua etapa – modificarea intensităților pixelilor utilizând formula de stretching pentru fiecare în parte

- Celelalte 5 thread-uri realizează operațiile specifice implementate în metoda run

4. Prezentare mod de utilizare, interacțiune cu utilizatorul, configurare

La o simplă rulare, aplicația cere utilizatorului să introducă de la tastatură o valoare a nivelului de stretching dorit, dar și a gradului de rotire a imaginii și poate să producă un mesaj de reintroducere a valorii dacă aceasta este „out of range” (în afara domeniului acceptat). Numele fișierului e citit în linie de comandă, aplicația deschide și citește fișierul cu acest nume, aplică operațiile de procesare și scrie alte 6 fișiere adaptate ca nume după original în directorul cu aceeași cale pe care o are și fișierul de intrare. Pe lângă fișierele de ieșire, utilizatorul va obține la output niște mesaje care sugerează timpii de execuție a procesărilor și modul în care are loc execuția (cum încep thread-urile, cum se termină, când

e scrisă imaginea procesată pe disc etc.). Utilizatorul trebuie să aibă instalate în prealabil un IDE pentru Java și un JDK adecvat (recomandări: Eclipse sau IntelliJ IDEA + JDK 8 sau 12).

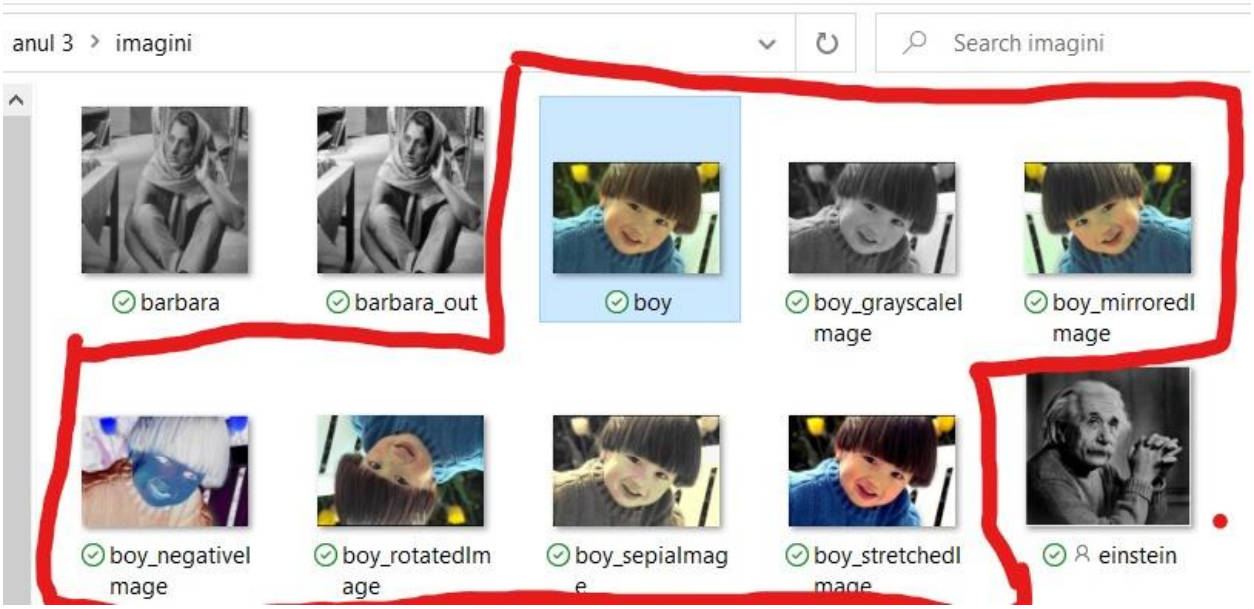


Figura 22: Un exemplu de rezultate așteptate după ce a fost rulată aplicația pentru imaginea sursă boy.bmp – 6 imagini procesate pe lângă imaginea originală

5. Concluzii

Din punct de vedere al performanței, aplicația înregistrează timpii de execuție pentru fiecare etapă în parte: citire imagine, construire histogramă, procesare imagine (etapa 1 + etapa 2) și scriere imagine pe disc.
Întrucât aplicația este implementată în paralel prin intermediul a 8 thread-uri, timpii de execuție sunt foarte mici, sub 1s, de unde rezultă performanțe ridicate în ceea ce privește timpul de execuție (utilizatorul realizează aproape instantaneu procesările după ce a introdus datele de la tastatură).

Exemplu la o rulare cu nivelul de stretching = 10% și grad de rotire = 180:

Alegeti valoarea nivelului de stretching ce va fi aplicat imaginii (valoare intreaga intre 0 si 50):
10
Alegeti gradul de rotire a imaginii (valoare intreaga intre 0 si 360):
180
Imaginea a fost citita in 10067 milisecunde
Starting Thread-0
Starting Thread-1
Starting Thread-2
Starting Thread-3
Starting Thread-4
Running Consumer
Starting Thread-5
Running Producer
Starting Thread-6
Starting Thread-7
Running rotating image thread, incepe procesarea imaginii
Running converting to grayscale thread, incepe procesarea imaginii
Running converting to negative image thread, incepe procesarea imaginii
Running converting to sepia image thread, incepe procesarea imaginii
Running image mirroring thread, incepe procesarea imaginii
Going dead rotating image thread, procesarea imaginii a fost efectuata
Going dead converting to grayscale thread, procesarea imaginii a fost efectuata
Going dead image mirroring thread, procesarea imaginii a fost efectuata
Going dead converting to negative image thread, procesarea imaginii a fost efectuata
Going dead converting to sepia image thread, procesarea imaginii a fost efectuata
Histograma RGB pentru imaginea data a fost creata in 179 milisecunde
Producer a pus partea 1/4 din histograma imaginii in buffer
Consumer a preluat partea 1/4 din histograma imaginii din buffer
Producer a pus partea 2/4 din histograma imaginii in buffer
Consumer a preluat partea 2/4 din histograma imaginii din buffer
Producer a pus partea 3/4 din histograma imaginii in buffer
Consumer a preluat partea 3/4 din histograma imaginii din buffer
Producer a pus partea 4/4 din histograma imaginii in buffer
Consumer a preluat partea 4/4 din histograma imaginii din buffer
Going dead Consumer
Etapa de consuming a durat 1476 milisecunde

Going dead Producer
Etapa de producing a durat 2253 milisecunde
Running stretching thread, incepe procesarea imaginii
Going dead stretching thread, procesarea imaginii a fost efectuata
Timpul pentru etapa de aflare a intensitatii maxime si minime a pixelilor pe fiecare canal R, G si B este de 128100 nanosecunde
Timpul pentru etapa de modificare a pixelilor imaginii folosind formula de stretching pentru fiecare in parte este de 228 milisecunde
Imaginile au fost procesate si scrise pe disc in 2594 milisecunde

Aplicația are o utilitate ridicată pentru utilizatorii care doresc o procesare a imaginilor foarte rapidă și sigură întrucât aplicația nu poate avea bug-uri și nu creează artefacte care să altereze calitatea imaginii după procesare. Ea este foarte simplă din punct vedere al utilizării și comunică rapid și interactiv cu utilizatorul.

Obiectivele aplicației sunt, așadar, următoarele :

- Realizarea unei imagini în nuanțe de gri folosind o imagine color
- Realizarea unei imagini cu efect Sepia folosind o imagine color
- Realizarea unei imagini negative folosind o imagine color
- Realizarea unei imagini cu un contrast ridicat specificat de către utilizator
- Realizarea unei imagini în oglindă folosind o imagine color
- Realizarea unei imagini rotite cu un anumit unghi folosind o imagine color

6. Referințe bibliografice

[1] **Wilhelm Burger și Mark J. Burge**, *Digital Image Processing. An algorithmic introduction using Java*, Editura Springer, New York, 2007

[2] **Bernd Girod**, *Digital Image Processing - Histograms Lectures*, Stanford University, 2013

[3] Curs despre procesarea digitală de imagini, Prof. Emmanuel Agu, Worcester Polytechnic Institute

[4] GeeksforGeeks

[5] Tech Vidvan

[6] Tutorials Point