Dunica David-Gabriel 333AA

Image Brightness Modification

1. Introducere

Proiectul pe care l-am abordat in cadrul acestei materii a fost modificarea luminozitatii unei imagini de format bitmap (.bmp) si apoi salvarea noii variante de poza. Atat numele pozei pe care o vom modifica cat si numele noii poze sunt parametri introdusi de la tastatura; pe langa acesti doi parametri se introduc de la tastatura si pathul spre poza si nivelul de brightness cu care se va modifica imaginea (parametru cuprins intre -100 si 100). Metoda implementeaza un model producer consumer paralel, in care producerul imparte matricea in 4 parti si le trimite consumerului care editeaza fragmentele cu noile valori ale pixelilor ca la finalul editarii sa compuna noua poza pe care sa o creeze.

2. Metoda de ediare a imaginii

Modificarea luminozitatii unei imagini se realizeaza prin cresterea, respectiv scaderea valorilor Red Green Blue ale fiecarui pixel. Acest lucru s-a putut realiza cu ajutorul bibliotecii java.awt.Color care faciliteaza convertirea pixelilor intr-un obiect de tip clasa care contine valori int pentru fiecare parametru RGB, astfel se lucreaza simplu pe fiecare pixel. Este de mentionat ca schimbarea luminozitatii se face prin adaugare sau scadere valori care in final trebuie sa ramana intre 0 si 255. Pentru a conserva aceast criteriu am creat o functie truncare care se asigura ca o eventuala depasire este transforata fie in minimul permis fie in maxim. Aceasta modificare a valorilor red green blue ale fiecarui pixel se face in metoda run din consumer, coloana cu coloana. Impartirea matricei de pixeli se face in metoda run din producer si este transmisa in consumer prin bufferul syncronized, astfel se pot preintampina probleme de deadlock intre cele doua threaduri pentru resursa partajata(bufferul).

Pasi:

- Transform imaginea intr-o matrice de pixeli
- Producerul o imparte in coloane si o trimite una cate una spre consumer.
- Consumerul primeste coloanele

- Folosesc metoda getRGB() pentru a imi impartii pixeli pe nivelul RGB
- Transform valorile RGB dupa urmatoarea formula
  CULOARE = truncate(CULOARE – VALOARE DE LA TASTATURA)
- Cream un element de tip Color cu noile valori (noul pixel)
- Cu metoda setRGB punem pixelul d-abia creat in poza
- Salvam noua poza in memoria calculatorului cu ajutorul bibliotecii ImageIO.

3. Descrierea modulelor utilizate

1. Clasa **Main** gazduieste metoda main; aceasta porneste programul in Java. De asemenea aaceasta clasa mosteneste clasa ThreadMaster. Clasa contine getters and setters pentru imagine + inaltime si latime. In cadrul functie main se verifica initial numarul de argumente introduse de la tastatura. Daca exista cele 4 se introduc in valorile necesare (path photoInput photoOutput si brightnessLevel), daca nu se introduc de la tastatura prin intermediul functiei inputRead() din clasa Input. Dupa introducerea acestor valori in clasa Input se formaza poza cu ajutorul metodei read din ImageIO si se seteaza valorile latimii si lungimii.
   In final se apeleaza functia threadMaster() pentru crearea threadurilor si apoi testam metodele abstracte si Varargs.
2. Clasa **ThreadMaster** mosteneste clasa Input. Am implementat o metoda ce numara nr de argumente (pentru varargs). Metoda threadMaster() utilizata in main da start si join la threadurile de producer/consumer. Pe scurt aceasta clasa se ocupa cu managementul celor doua threaduri create si le faciliteaza memoria partajata de tip buffer (Clasa Buffer) in care am avut grija sa tratam problema de tip deadlock in care ambele threaduri asteapta dupa celalalt la nesfarsit.
3. Clasa abstracta **Input** are ca scop stocarea informatiilor date ca input. Metoda inputRead() se ocupa de citirea de la tastatura; se creaza un obiect de tip scanner cu ajutorul caruia vom citi de la tastarura mai intai pathul spre poza , apoi numele pozei pe care o vom edita, apoi

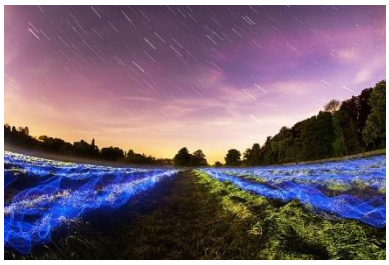
numele pozei pe care o vom crea in urma editarii si in final se citeste o valoare cuprinsa intre -100 si 100 care reprezinta nivelul de Brightness cu care va fi editata poza. Clasa implementeaza Interface. Am mai implementat o metoda abstracta din interfata. Este de mentionat ca aceasta clasa cuprinde si setters si getters pentru cele patru variabile citite de la tastatura.

4. Interfata **Interface** contine o metoda pe care am implentat-o in clasa Input.
5. Clasa **Producer** este o extindere a clasei Main si implementeaza Runnable. Este o clasa care se executa cu Java multi Threads. Producerul se ocupa cu citirea imaginii intr-o matrice de pixelica care se trimite prin bufferul creat cu clasa Buffer in 4 parti ale imaginii in clasa Consumer. In metoda run se citeste cate un sfert din poza pe rand; acest lucru este facilitat de doua variabile start si end care tin contorul inceputului si finalului zonei din care se va extrage portiunea de imagine. Dupa citirea fiecarui sfert aceasta parte este trimisa pe buffer pentru a fi luata de consumer. Aceasta operatiune este monitorizata din punct de vedere al timpului de executie (in final se afiseaza timpul in care s-au transmis toate coloanele prin buffer).
6. Clasa **Consumer** este asemanatoare clasei Producer. Aceasta extinde de asemenea Mainul si implementeaza Runnable. Clasa Consumer primeste prin buffer cele 4 sferturi de imagine pe care le modifica pixel cu pixel prin metoda explicata la punctul 2. Dupa ediarea coloanelor se formeaza o noua matrice care se compune in finalul executiei in fisierul de output prin biblioteca ImageIO. In metoda run intial se primeste prin buffer sfert cu sfert de la Producer matricea (poza in format matricial cu valorile pixelilor). Dupa aceasta operatiune se afiseaza timpul in care s-au primit datele prin buffer. In cele ce urmeaza s-a parcurs matricea linie-coloana si s-au editat pixel cu pixel valorile rgb dupa metoda descrisa mai sus. Dupa finalizarea acestei operatii se afiseaza timpul de modificare a valorilor pixelilor, iar in final se creaza noua poza cu ajutorul matricii create mai sus. De asemenea acest proces este monitorizat din punct de vedere al timpului.

7. Clasa **Buffer** extinde de asemenea clasa Main. Clasa contine doua metode syncronized care elimina riscul de Deadlock (resursele regasite in aceste doua metode au acces concurent => accesul la buffer se face pe rand de catre threaduri). Aceasta clasa contine doua metode: una de put si una de get. Care in mod instinctiv scot si baga date prin buffer. Este de montinat ca fiind o memorie partajata intre doua threaduri inseamna ca pot aparea probleme de sincronizare: dupa ce producerul pune d-abia atunci consumerul poate sa scoata, dupa ce consumarul scoare, d-abia atunci poate iar producerul sa puna si tot asa. Astfel cele doua metode sunt syncronized si au o variabila transfer care daca este true permite folosirea memoriei de catre producer, iar daca este false se foloseste de catre consumer. De asemenea aceste metode au un contor care afiseaza dupa fiecare operatie coloana care a fost fie pusa fie scoasa de pe buffer. Astfel se poate observa foarte clar functionalitatea regiunii critice => imediat dupa ce se pune pe buffer se scoate .. si tot asa.

4. Rezulate si timpii obtinuti

- Test 1 rezolutie (640x426)



Dunica David-Gabriel 333AA

```
<Columns Send Time>----------------------<
Consumer recive 4 part of image
  -> Start time: 1644915654351 ms
  -> Final time: 1644915655111 ms
  -> Working time = 760 ms
----------------------------------------<

<Columns Recive Time>--------------------<
  -> Start time: 1644915654351 ms
  -> Final time: 1644915655238 ms
  -> Working time = 887 ms
----------------------------------------<

<Columns Modify Time>--------------------<
  -> Start time: 1644915655239 ms
  -> Final time: 1644915655316 ms
  -> Working time = 77 ms
----------------------------------------<

<Columns Write Time>---------------------<
  -> Start time: 1644915655316 ms
  -> Final time: 1644915655346 ms
  -> Working time = 30 ms
----------------------------------------<
```

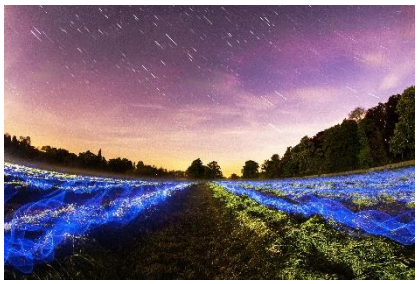
```
<Columns Send Time>----------------------<
Consumer recive 4 part of image
  -> Start time: 1644915577219 ms
  -> Final time: 1644915577981 ms
  -> Working time = 762 ms
----------------------------------------<

<Columns Recive Time>--------------------<
  -> Start time: 1644915577219 ms
  -> Final time: 1644915578104 ms
  -> Working time = 885 ms
----------------------------------------<

<Columns Modify Time>--------------------<
  -> Start time: 1644915578105 ms
  -> Final time: 1644915578179 ms
  -> Working time = 74 ms
----------------------------------------<

<Columns Write Time>---------------------<
  -> Start time: 1644915578179 ms
  -> Final time: 1644915578209 ms
  -> Working time = 30 ms
----------------------------------------<
```

- Test 2 rezolutie (5184x3456)

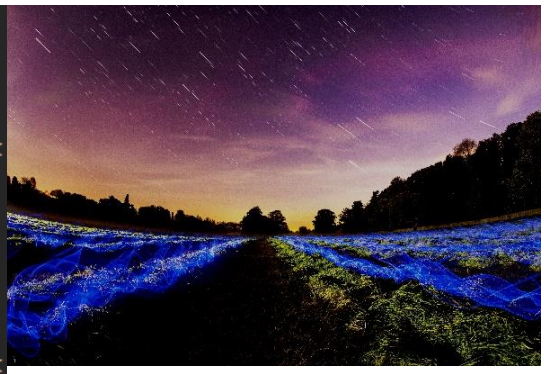


```
<Columns Send Time>----------------------<
  -> Start time: 1644915713968 ms
  -> Final time: 1644915715005 ms
  -> Working time = 1037 ms
-----------------------------------------<
Consumer recive 4 part of image

<Columns Recive Time>--------------------<
  -> Start time: 1644915713968 ms
  -> Final time: 1644915715460 ms
  -> Working time = 1492 ms
-----------------------------------------<

<Columns Modify Time>--------------------<
  -> Start time: 1644915715461 ms
  -> Final time: 1644915716208 ms
  -> Working time = 747 ms
-----------------------------------------<

<Columns Write Time>---------------------<
  -> Start time: 1644915716208 ms
  -> Final time: 1644915716545 ms
  -> Working time = 337 ms
-----------------------------------------<
```



- Test 3 rezolutie (512x512)

```
<Columns Send Time>----------------------<
Consumer recive 4 part of image
  -> Start time: 1644915821521 ms
  -> Final time: 1644915822278 ms
  -> Working time = 757 ms
-----------------------------------------<

<Columns Recive Time>--------------------<
  -> Start time: 1644915821521 ms
  -> Final time: 1644915822400 ms
  -> Working time = 879 ms
-----------------------------------------<

<Columns Modify Time>--------------------<
  -> Start time: 1644915822401 ms
  -> Final time: 1644915822467 ms
  -> Working time = 66 ms
-----------------------------------------<

<Columns Write Time>---------------------<
  -> Start time: 1644915822467 ms
  -> Final time: 1644915822502 ms
  -> Working time = 35 ms
-----------------------------------------<
```

5. Concluzii

Din punct de vedere al functionalitati se poate observa din testele executate mai sus ca algoritmul functioneaza atat la marirea luminozitatii cat si la scaderea acesteia. Testul 1 s-a executat cu o crestere si o scadere a nivelului de luminozitate de 80, testul 2 s-a executat cu o scadere a luminozitatii de 70 si testul 3 s-a executat cu o crestere a luminozitatii de 50.

Se poate sesiza cresterea semnificativa de timp intre testele 1 si 2. Acest lucru este datorat dimensiunii: a doua poza (in format 4k) a trebuit sa execute mult mai multe operatii. Avand in vedere ca transmisia se face in patru parti adevarata modificare temporala intre cele doua teste intervinde de abia la partea de prelucrare a datelor; diferenta dintre cele doua etape de trimitere este semnificativ mai mica fata de etapa de prelucrare, care dureaza de 10 ori mai mult. De asemenea etapa de creare a noii imagini dureaza de 10 ori mai mult in cazul testului 2.

O alta remarca este diferenta foarte mica dintre timpul de trimitere si timpul de primire de pe buffer. Acest fenomen arata corectitudinea de executare a problemei consumer producer. Timpul de primire fiind f putin mai mare fata de timpul de trimitere. Acest lucru este datorat faptului ca consumerul trebuie sa astepte dupa producer. Deci dupa ce producerul trimite ultima coloana prin buffer consumerul inca asteapta semnalul de a putea sa o ia de pe buffer.

O ultima remarca referitoare la diferenta intre modificarea luminazitatii in sus sau in jos. Nu s-a gasit nicio diferenta sesizanta in cresterea de timp in testul 1. Atunci cand am testat atat cresterea de luminozitate cat si scaderea ei pe aceasi poza.

6. Bibliografie

- https://www.geeksforgeeks.org/java-program-to-increase-or-decrease-brightness-of-an-image/