POLYTECH
NICE-SOPHIA

Projet:

Traitement d'images en java

Réalisé par : -Nada BENCHEKROUN
-Adam BENMOULOUD

Encadré par : -Lopez PACHECO
-Marie GUYOMARD

2020/2021

POLYTECH
NICE-SOPHIA

SOMMAIRE:

POLYTECH
NICE-SOPHIA

Introduction:

Ce rapport a pour but de décrire le déroulement de notre projet JAVA en MAM3. Il contient l'ensemble des éléments du projet décrits dans le sommaire. Ce projet en traitement d'images consiste principalement à traiter une image donnée en entrée par l'utilisateur et d'y appliquer différents filtres.

Pour une image en échelle de gris, l'utilisateur peut faire une analyse de l'exposition pour voir graphiquement l'histogramme, exécuter un traitement d'assombrissement et éclairage, effectuer un traitement local par une matrice de convolution.

Pour une image couleur (formaté sur 4 canaux -i.e. RGBα-), on a l' option de transformation en échelle de gris, analyser l'exposition de chaque canal RGB et d'exécuter un traitement d'assombrissement et éclairage.

Dans ce rapport nous parlerons tout d'abord du point de vue technique; nous présenterons le fonctionnement des différentes classes dans son ensemble ainsi que les éléments qui prouvent le bon fonctionnement de celui-ci. Dans un second temps, nous présenterons nos impressions sur le projet concernant les difficultés techniques rencontrées et les perspectives ouvertes.

Nous espérons que vous prendrez autant de plaisir à lire ce rapport que nous en avons pris durant tout le déroulement de ce projet.

POLYTECH
NICE-SOPHIA

1. Installation : Etapes d'excécution et compilation de TestTraitementImage

Pour accéder au programme, l'utilisateur doit décompresser le fichier benmouloud-benchekroun.zip. Il aura donc ensuite accès au fichier benmouloud-benchekroun qui contient les fichiers TestTraitementImage.java, README.pdf et benmouloud_benchekroun.jar.

L'utilisateur peut lancer le programme de deux manières :

- Il peut utiliser le fichier .jar sous sa forme non exécutable. Pour faire cela, il devra ajouter au dossier les librairies jai_codec.jar, jai_core.jar et mlibwrapper_jai.jar. L'utilisateur pourra ensuite ouvrir la console Linux dans ce dossier et compiler le fichier TestTraitementImage.java grâce à cette commande :

  javac -cp .:./benmouloud_benchekroun.jar TestTraitementImage.java

  Après avoir compilé le fichier, on peut utiliser la commande suivante pour lancer le programme : **java -cp .:./benmouloud_benchekroun.jar TestTraitementImage**

  D'autres manipulations sont possibles comme en modifiant la variable d'environnement CLASSPATH et bougeant le package là où l'on stocke tous les autres packages. Cependant, cette méthode a l'avantage d'être rapide de montrer que nous avons réussi à modifier le fichier MANIFEST.MF.

- Il peut aussi utiliser le fichier .jar sous sa forme exécutable. Pour faire cela, l'utilisateur devra ouvrir la console Linux dans ce dossier et utiliser la commande :

  java -Xboothclasspath/a:/CHEMIN/jai_codec.jar:/CHEMIN/jai_core.jar:/CHEMIN/mlibwrapper_jai.jar -jar benmouloud_benchekroun.jar

  Avec **/CHEMIN/** qui correspond au chemin pour accéder aux librairies de l'utilisateur. Dans notre cas, **/CHEMIN/** = **/home/user/myjavalibs/** .

2. Utilisation : Présentation des classes et du menu de l'application

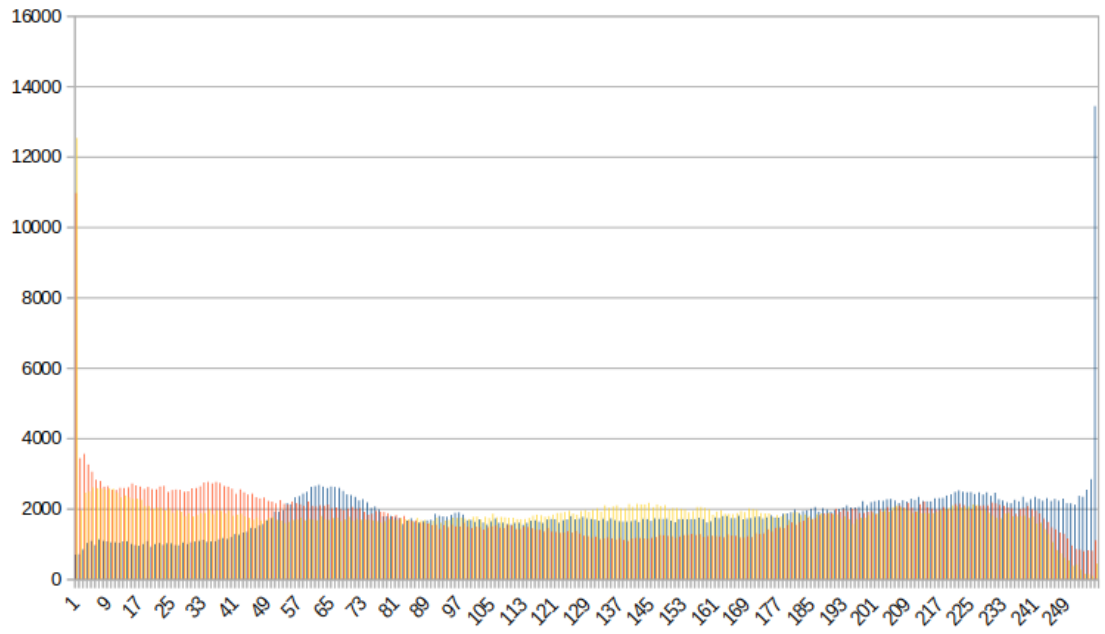
- **Les Classes**

-**Classe TraitementImage.java** : la Classe Mère du projet

-**Classe TraitementImageCouleur.java** : classe qui hérite de la classe mère TraitementImage.java et s'occupe principalement du traitement et de l'analyse des images couleurs pour en appliquer des filtres ou faire une analyse. Les traitement possibles sont les suivants:

- ➔ l'option de transformation en échelle de gris : méthode public void conversionGrayscale()
- ➔ l'option d'exécuter un traitement éclairage : méthode public void eclaircir()
- ➔ l'option d'exécuter un traitement d'assombrissement : méthode public void assombrir()
- ➔ l'option d'analyser l'exposition de chaque canal RGB pour avoir nos histogrammes : public void analyse()

  Analyse appliqué à **the-legend.png :** On peut voir ci-dessous graphiquement l'histogramme en important le fichier en Excel



-Classe TraitementImageNiveauGris.java : classe qui hérite de la classe mère TraitementImage.java et s'occupe principalement du traitement et de l'analyse des images couleurs pour en appliquer des filtres ou faire une analyse. Les traitement possibles sont les suivants:

- ➔ l'option d'exécuter un traitement d'éclairage : méthode public void eclaircir()
- ➔ l'option d'exécuter un traitement d'assombrissement : méthode public void assombrir()
- ➔ l'option d'analyser l'exposition pour avoir nos histogrammes : public void analyse()
- ➔ l'option d'effectuer un traitement local par une matrice de convolution : méthode public void traitementLocal

-Classe InterfaceUtilisateur.java : classe pour interagir avec l'utilisateur pour demander

par exemple, le type de traitement à effectuer sur une image. Elle contient le menu de l'application qui sera affiché pour l'utilisateur au terminal pour effectuer le type de traitement ou analyse souhaité.

-**Classe TestTraitementImage.java** : Pour répondre au besoin de traitement de l'utilisateur, la classe TestTraitementImage fera appel à la classe InterfaceUtilisateur pour compiler le projet.

- **<u>Menu de l'application Traitement d'image:</u>**

  Le menu de notre application traitement image est explicite dans la classe InterfaceUtilisateur.java. Il propose divers traitements filtres et analyses à faire pour l'utilisateur.

  De prime abord, l'utilisateur est amené à entrer le nom du fichier à traiter sans l'extension, exemple : **the-legend**, puis d'entrer l'extension du fichier à traiter, (**.png**) , ensuite le chemin pour accéder au fichier dans notre cas : **./image/**

  Au final tout le menu de notre application s'affiche pour l'utilisateur.

----------Menu:----------

- **AnalyseExp (ou ana)** : : pour faire une analyse de l'exposition de l'image,
- **Grayscale (ou g)** : pour exécuter une transformation en échelle de gris de l'image,
- **Assombr (ou asm)** : pour exécuter un traitement d'assombrissement sur l'image,
- **Ecl (ou ec)** : pour exécuter un traitement d'eclairage sur l'image,
- **Sortir(ou S):** pour quitter.

```
user@user-VirtualBox:~/myjavalibs/mam3/ipa/projet$ javac TestTraitementImage.java
user@user-VirtualBox:~/myjavalibs/mam3/ipa/projet$ java TestTraitementImage
Entrez le nom du fichier à traiter (sans extension) :
> the-legend
Entrez l'extension de ce fichier à traiter (exemple : .png) :
> .png
Entrez le chemin pour accéder au fichier à traiter (format .../dossier/) :
> ./images/
--------  # Menu:      ------------------------------------------------------------------
Tapez une des commandes suivantes pour exécuter un traitement ou une analyse sur l'image,
tapez 'sortir' pour quitter l'application.

AnalyseExp (ou ana) :         pour faire une analyse de l'exposition de l'image,

Grayscale (ou g)  :         pour exécuter une transformation en échelle de gris de l'image,

Assombr (ou asm)  :         pour exécuter un traitement d'assombrissement sur l'image,

Ecl (ou ec)       :         pour exécuter un traitement d'eclairage sur l'image,

Sortir(ou S):    pour quitter.

-------------------------------------------------------------------------------------

>
```

3. ToDo : Explications du fonctionnement du programme

quelques précisions:

-Notre application Java traitement d'image, l'utilisateur est amené à entrer une image en gris ou en couleur pour effectuer tous les traitements et analyses possibles. Cependant, il n'accepte pas de fichiers image autre que les fichiers - PNG- , le programme ne prend en compte que les -PNG-.

-Notre application propose à l'utilisateur un traitement ou analyse de chaque image couleur ou image en gris -grayscale-. Pour une **image en échelle de gris**, l'utilisateur peut faire une analyse de l'exposition pour voir graphiquement l'histogramme, exécuter un traitement d'assombrissement et éclairage, effectuer un traitement local par une matrice de convolution. En effet, pour **une image couleur** (formaté sur 4 canaux -i.e. RGBα-), on a l' option de transformation en échelle de gris, analyser l'exposition de chaque canal RGB et d'exécuter un traitement d'assombrissement et éclairage.

-Il n'est pas possible de changer l'image que l'utilisateur a entré pour le traitement qu'à la fin; en sélectionnant sortir (ou s) l'utilisateur peut à cet effet relancer le programme et effectuer un nouveau traitement ou analyse de l'image de son choix.

-On précise que notre application traite la *partie 1*: 12/20 et la *partie 2*: 18/20 et non pas la *partie 3*: 20/20 qu'on a pas pu traiter, afin de peaufiner et de rendre un programme robuste et fonctionnel traitant l'intégralité des deux premières parties.

4. Les Algorithmes:

- **Classe TraitementImage :**

  - **Constructeur :** On prend en paramètre le chemin vers le fichier, son nom et son extension. Ces paramètres seront rentrées dans les variables de classe correspondantes. On créera ensuite une BufferedImage, on rentrera aussi les dimensions de l'image dans les variables de classe correspondantes.

    Après avoir créé un objet ColorModel, on vérifie le type de l'image (niveau de gris ou couleur) pour remplir le tableau correspondant : tableau byte si en niveau de gris et tableau int si en couleur.
  - **Méthodes Setters et getters** : Ces méthodes sont utilisées pour utiliser et modifier les divers variables de la classe TraitementImage.

- **Classe TraitementImageNiveauGris : hérite de la classe mère TraitementImage**

  - **Constructeur :** On utilise le constructeur que l'on hérite de la classe mère grâce à super. Elle a comme argument String cheminFichier, String nomFichier et String extensionFichier.

  - **Méthode eclaircir() :** On parcourt le tableau de type byte pour modifier chaque pixel. On éclaircit l'image en calculant la racine carrée de la valeur du byte (entre 0 et 255), puis on normalise cela avant de remplacer l'ancienne valeur par la nouvelle. Enfin, on crée une nouvelle image à partir du tableau byte modifié. Elle aura comme nom "nomFichier-eclairee".

  - **Méthode assombrir() :** On parcourt le tableau de type byte pour modifier chaque pixel. On assombrit l'image en calculant le carré de la valeur du byte (entre 0 et 255), puis on normalise cela avant de remplacer l'ancienne valeur par la nouvelle. Enfin, on crée une nouvelle image à partir du tableau byte modifié. Elle aura comme nom "nomFichier-assombrie".
  - **Méthode traitementLocal() :** On demande à l'utilisateur de rentrer le nom du fichier csv qui correspond à la matrice de convolution que l'on veut utiliser pour le traitement local. Après cela, on parcourt le tableau de type byte pour modifier chaque pixel. On calcule la valeur correspondant au pixel en fonction des pixels voisins grâce à la matrice de convolution. Les cas particuliers (les bords) sont fait de manière indépendante en fonction des nombres qui ne sont pas utilisables dans la matrice de convolution. On normalise par la suite chacun de ces résultats. Enfin, on crée une nouvelle image à partir du tableau byte modifié. Elle aura comme nom "nomFichier-traiteeLocalement".

  - **Méthode analyse() :** Pour toutes les valeurs entre 0 et 255, on parcourt le tableau de type byte pour obtenir chaque pixel. On regarde ensuite si l'une de ces valeurs y est égale. Si c'est le cas, on incrémente une fois le nombre d'occurrence de cette valeur. On

obtient donc à la fin un tableau de type int de dimension 2*256. La première colonne correspond à toutes les valeurs possibles entre 0 et 255. L'autre quant à elle correspond au nombre d'occurrences de ces valeurs. Pour finir, on crée un fichier .txt où l'on réécrit les valeurs qui sont dans ce nouveau tableau. Il aura comme nom "nomFichier-h".

- **Méthode normalized(int in, int maxValue) :** On utilise cette méthode pour normaliser les résultats des calculs de cette classe. On normalise pour avoir des valeurs entre 0 et 255.

- **Classe TraitementImageCouleur : hérite de la classe TraitementImage**

  - **Constructeur :** On utilise le constructeur que l'on hérite de la classe mère grâce à super. Elle a comme argument String cheminFichier, String nomFichier et String extensionFichier.

  - **Méthode eclaircir() :** On parcourt le tableau de type int pour modifier chaque pixel. On extrait de chacun des int, les valeurs entre 0 et 255 correspondant à R, G et B. On éclaircit l'image en calculant la racine carrée de chacune des valeurs de R, G et B que l'on normalise ensuite avant créer un nouvel objet Color grâce à ces 3 valeurs. On extrait de cette dernière la nouvelle valeur int. On l'utilise pour remplacer l'ancienne valeur dans le tableau int. Enfin, on crée une nouvelle image à partir du tableau byte modifié. Elle aura comme nom "nomFichier-eclairee".

  - **Méthode assombrir() :** On parcourt le tableau de type int pour modifier chaque pixel. On extrait de chacun des int, les valeurs entre 0 et 255 correspondant à R, G et B. On éclaircit l'image en calculant le carré de chacune des valeurs de R, G et B que l'on normalise ensuite avant créer un nouvel objet Color grâce à ces 3 valeurs. On extrait de cette dernière la nouvelle valeur int. On l'utilise pour remplacer l'ancienne valeur dans

le tableau int. Enfin, on crée une nouvelle image à partir du tableau byte modifié. Elle aura comme nom “nomFichier-assombrie”.

- **Méthode conversionGreyscale() :** On parcourt le tableau de type int pour obtenir chaque pixel. On extrait de chacun des int, les valeurs entre 0 et 255 correspondant à R, G et B. On en fait la moyenne puis on les met dans un nouveau tableau de type byte. Après cela, on crée cette fois-ci une nouvelle image en niveau de gris. Elle aura comme nom “nomFichier-grayscale”.

- **Méthode analyse() :** Pour toutes les valeurs entre 0 et 255, on parcourt le tableau de type int pour obtenir chaque pixel. On extrait de chacun des int, les valeurs entre 0 et 255 correspondant à R, G et B. On regarde ensuite si l’une de ces valeurs y est égale. Si c’est le cas, on incrémente une fois le nombre d'occurrence de cette valeur pour cette colonne (R, G ou B). On obtient donc à la fin un tableau de type int de dimension 4*256. La première colonne correspond à toutes les valeurs possibles entre 0 et 255. Les trois autres quant à elles correspondent au nombre d'occurrences de ces valeurs en R, G et B. Pour finir, on crée un fichier .txt où l’on réécrit les valeurs qui sont dans ce nouveau tableau. Il aura comme nom “nomFichier-h”.

- **Méthode normalized(int in, int maxValue) :** On utilise cette méthode pour normaliser les résultats des calculs de cette classe. On normalise pour avoir des valeurs entre 0 et 255.

- **Classe InterfaceUtilisateur :**

  - **Méthode estCouleur(RenderedOp ropimage) :** On crée une BufferedImage puis l'on regarde s'il s'agit d'une image en niveau de gris ou en couleur. S'il s'agit d'une image en couleur on renvoie le booléen true. Sinon, on renvoie le booléen false.

  - **Méthode afficherMenu(Boolean isCouleur) :** On affiche le menu des options disponibles dans le programme en fonction de s'il s'agit d'une image en couleur ou non.

  - **Méthode lancerMenu() :** On demande à l'utilisateur d'entrer les détails du fichier : le nom (sans extension), l'extension puis le chemin pour y accéder. On vérifie ensuite si l'objet existe, si non on arrête le programme. Si oui, on créer une boucle while qui s'arrêtera quand l'utilisateur aura rentrer s ou sortir dans le terminal.

    On affiche le menu en fonction du type de l'image (couleur ou niveau de gris). En fonction de ce que l'utilisateur frappe dans le terminal, on crée un objet TraitementImageNiveauGris ou TraitementImageCouleur pour y appliquer le traitement souhaité.

    Lorsque l'utilisateur frappe sortie ou s, la boucle et le système s'arrêtent.

5. Difficultés et perspectives ouvertes

Ce projet nous a permis d'utiliser toutes les connaissances acquises durant le premier semestre de MAM3: création de différentes classe et objets, utilisation de boucles (while, for, switch), utilisations des packages, l'héritage, les exceptions (try,catch)....

D'autres concepts ont été appris en autonomie, comme travailler avec la librairie JAI « Java Advanced Imaging » qui fournit une API pour manipuler des images. Nous l'avons trouvé difficilement appréhendable au début du travail.

L'une des difficultés majeures de ce projet a été de trouver des concepts (non vus en cours) qui soient adaptés à nos problématiques et qui soient simples à coder tout comme à mettre en œuvre. En Java, il existe des lignes de codes très différentes, de toute sorte de difficultés, plus ou moins longues mais qui peuvent aboutir au même résultat.

La répartition des tâches fut aussi l'un des aspects de ce projet que nous avons le plus redouté. En effet, chacun avait des parties assignées à finir avant un certain temps. L'objectif était de faire des réunions Zoom régulières pour faire le point sur ce que nous avons fait chacun de notre côté et pour se donner de nouveaux objectifs individuels. Au bout du compte, cette organisation a été un succès et nous a permis de répartir le travail équitablement et d'arriver à nos objectifs dans le temps imparti.

Pour finir, il y a une perspective possible qui nous est assez claire. Nous aurions pu traiter la partie 20/20 proposant un traitement local par la matrice de convolution sur chaque canal RGBα avec normalisations. Cependant par contrainte de temps, nous avons décidé de ne pas aborder cette partie afin de bien peaufiner les deux premières parties de notre projet.

Conclusion:

De l'avis général, à travers ce premier projet en java nous avons consolidé toutes nos connaissances acquises pendant le semestre en programmation orienté objet. De plus, ce projet nous a fait découvrir de nouveaux concepts intéressants.

Nous sommes globalement satisfaits de ce que nous avons réalisé car nos principaux objectifs ont été atteints dans les délais et notre application est bien robuste et fonctionnelle

Une très bonne expérience !