R5.06 - Traitement de l'image

TP2 : Manipulation d'Image

Contenu de la Page

Nous allons dans ce TP, écrire différents algorithmes de transformation d'images.

Nous avons mis en avant dans le TP précédent, que nous pouvions avoir des images chargées sur des composants graphiques, et des images uniquement chargées en mémoire sur notre controleur.

Java gère différents types d'images, **nous allons principalement nous concentrer sur des images PNG**.

Exercice 1 : Affichage du détail d'une couleur

retour haut de page

Un pixel est constitué de 3 valeurs RGB (Red Green Blue), plus d'une couche alpha pour spécifier si le pixel est transparent ou opaque. En raison de la présence de cette couche alpha nous devons faire un décalage de bits, pour récupérer la valeur entière d'une couleur.

Nous avons vus dans le TP précédent qu'il était possible de récupérer la valeur entière d'une couleur.

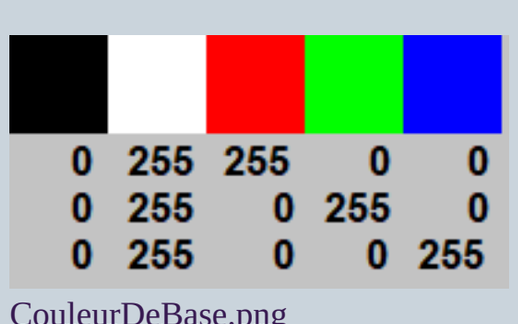
Cette valeur entière est obtenue à l'aide de la formule :

rouge * 256*256 + vert * 256 +bleu.

Dans notre langage, on peut facilement, à partir de la couleur entière, récupérer les différentes valeurs de **Rouge**, **Vert**, **Bleu**. Il suffit de passer par une couleur intermédiaire de Type Color.

Pour cet exercice, nous n'allons pas utiliser la classe Color, mais décomposer l'entier par divisions successives pour récupérer les quantités de Rouge, Vert, Bleu .

Nous souhaitons dans cet exercice récupérer les valeurs RGB de différentes couleurs de carrés stockées dans une image.



CouleurDeBase.png

Nous vous fournissons pour cela une classe CouleurDeBase

Voici la trace d'exécution attendue :

```
coulCarreNoir  :           0 [  0,  0,  0]
coulCarreBlanc :  16 777 215 [255,255,255]
coulCarreRouge :  16 711 680 [255,  0,  0]
coulCarreVert  :      65 280 [  0,255,  0]
coulCarreBleu  :         255 [  0,  0,255]
```

Travail à Faire

1 Récupérez les fichiers CouleurDeBase.png et CouleurDeBase.java

Ecrire le code de la classe **ImageUtil**, permettant d'obtenir le résultat attendu.
Théoriquement, vous n'avez pas besoin de modifier le code de la classe CouleurDeBase.

Exercice 2 : Transformation en niveau de gris

retour haut de page

Lorsque les valeurs de Rouge, Vert, et Bleu sont identiques, nous obtenons du gris.
Lorsque les valeurs RGB sont égales à 0 nous avons du Noir.
Lorsque les valeurs RGB sont égales à 255, nous avons du Blanc.

Entre les deux nous obtenons des gris allant du gris foncé au gris clair.

Il existe plusieurs algorithmes pour transformer un pixel de couleur en un niveau de gris.

Nous vous proposons d'aborder trois algorithmes différents.

Algo 1
On détermine le min et le max des trois valeurs RGB, puis on fait la moyenne de ces deux valeurs.

Algo 2
On fait la moyenne des trois valeurs RGB.

Algo 3
On calcule la luminance (ou intensité) du pixel en utilisant une somme pondérée des trois valeurs RGB.

Rouge ayant pour coefficient 0.299
Vert ayant pour coefficient 0.587
Bleu ayant pour coefficient 0.114

Voici un exemple d'image transformée :

asterix_couleur.png

Travail à Faire

2.1 Ajouter dans la classe ImageUtil, une méthode `int luminance (Color coul, int numAlgo)`
Qui retourne la luminance d'une couleur en appliquant la règle de l'algo `num_algo` choisi.

2.2 Créer un programme ConvertNoirEtBlanc, qui devra se lancer avec la ligne de commande :
`java ConvertNoirEtBlanc fichierSource.png fichierDest.png [numAlgo]`.
Par défaut le troisième paramètre numAlgo vaudra 1.

Vous devrez vérifier :
que les noms des deux fichiers images sont bien renseignés et se termine par `.png`
que si le numéro d'algo est renseigné, il devra appartenir à l'intervalle [1;3]

Lancer trois fois le programme conserver précieusement les images pour un prochain exercice.

Exercice 3 : Luminosité et contraste

retour haut de page

Luminosité

Eclaircir une image consiste à augmenter du même nombre d'unité chaque valeur Rouge Vert Bleu.
Assombrir une image consiste à diminuer du même nombre d'unité chaque valeur Rouge Vert Bleu.

Théoriquement on pourrait faire varier les unités de -255 à +255

Contraste

Il existe plusieurs algorithmes pour modifier le contraste. Nous allons pour cette exercice retenir une seule méthode

Une valeur appartient à l'intervalle [0;255], nous définirons comme `valeur_moyenne` le milieu de cet intervalle, soit 127.

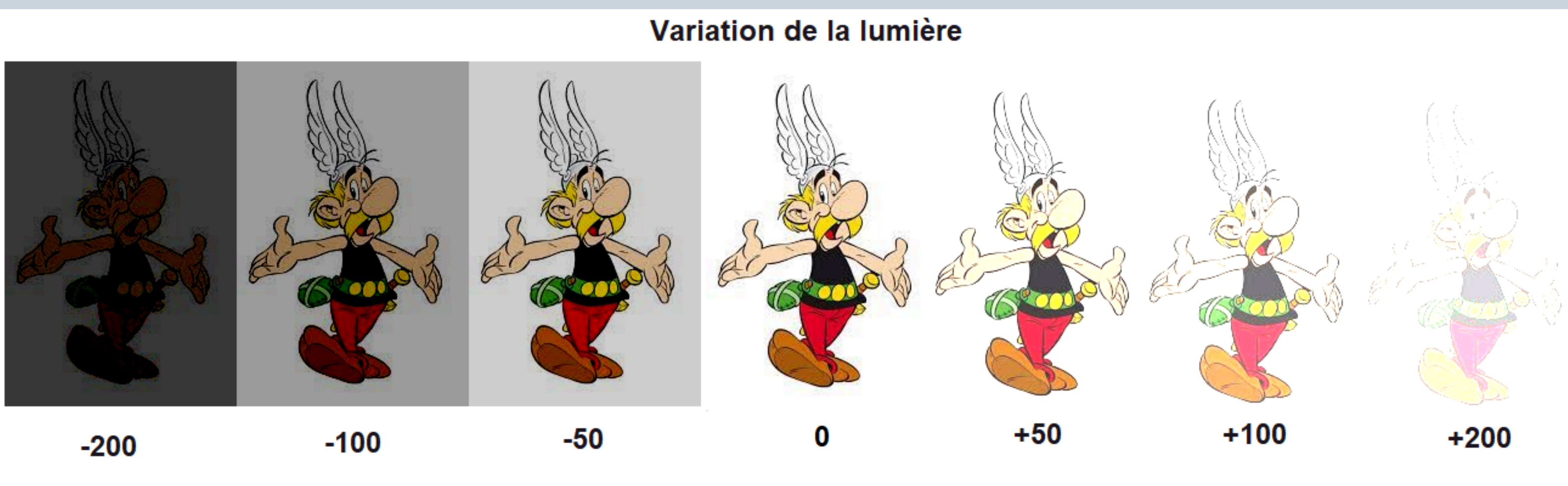
Quand on **augmente** le contraste, on **écarte** la valeur de la `valeur_moyenne`.
Ainsi si la valeur est < 127, la valeur diminue, sinon la valeur augmente.

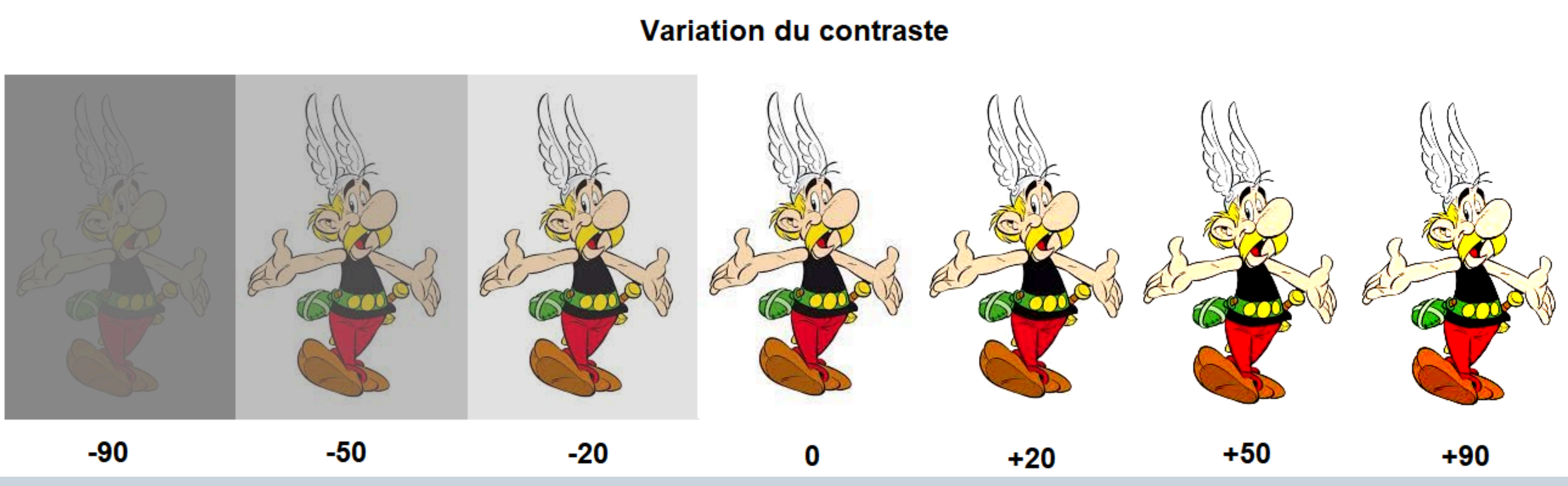
Quand on **diminue** le contraste, on **rapproche** la valeur de la `valeur_moyenne`.
Ainsi si la valeur est < 127, la valeur augmente, sinon la valeur diminue.

La formule pour modifier chaque valeur est la suivante :

nouvelle valeur = valeur + contraste / 100 x (valeur-127)

contraste pouvant varier de -100 à +100





Travail à Faire

3.1 Ecrire un programme Lumiere, permettant de modifier la luminosité d'une image.

Le programme devra se lancer grâce à la commande suivante :
`java Lumiere fichierSource.png fichierDest.png valeur`

3.2 Ecrire un programme Contraste permettant de modifier le contraste d'une image.

Le programme devra se lancer grâce à la commande suivante :
`java Contraste fichierSource.png fichierDest.png valeur`

Exercice 4 : Pot de peinture

retour haut de page

Nous allons dans cet exercice réaliser un outil pot de peinture qui permettra à partir d'un point d'origine, de remplacer ce point et surtout tous les points adjacents **en cascade** de cette couleur par une autre couleur.

Mais pour commencer nous allons mettre au point la mécanique à partir d'un tableau de caractère. Soit le tableau de 11 par 9 suivant :

	0	1	2	3	4	5	6	7	8
0									
1									
2				X	X	X			
3			X				X		
4		X					X		
5		X					X		
6			X	X	X		X		
7						X			
8									
9									
10									

Nous souhaitons colorier avec des **o** toute la zone centrale délimitée par des X.

Nous allons pour cela définir une méthode peinture (int lig, int col, char coul).

lig et col représentent les coordonnés du point de départ (exemple 4,3).
coul représente la couleur qu'il faut appliquer (dans notre exemple ce sera un **o**)

Cette méthode va identifier la couleur (ici le caractère ' ') du point d'origine, puis appellera une méthode peintureRec (int lig, in col, char coulOrig, char coulDest)
qui affectera dans le point d'originie la couleur coulDest (pour nous un caractère), et qui appellera récursivement peintureRec pour :
- Le point situé au dessus.
- Le point situé à gauche.
- Le point situé en dessous.
- Le point situé à droite.

vous devriez obtenir le resultat suivant:

	0	1	2	3	4	5	6	7	8
0									
1									
2				X	X	X			
3			X	o	o	o	X		
4		X	o	o	o	o	X		
5		X	o	o	o	o	X		
6			X	X	X	o	X		
7						X			
8									
9									
10									

Travail à Faire

4.1 Ecrire un programme permettant de tester vos méthodes peinture et peintureRec
4.2 Testtez maintenant en sélectionnant le point d'origine (9,4). Vous devriez obtenir le résulat suivant:

	0	1	2	3	4	5	6	7	8
0	o	o	o	o	o	o	o	o	o
1	o	o	o	o	o	o	o	o	o
2	o	o	o	X	X	X	o	o	o
3	o	o	X				X	o	o
4	o	X					X	o	o
5	o	X					X	o	o
6	o	o	X	X	X		X	o	o
7	o	o	o	o	o	X	o	o	o
8	o	o	o	o	o	o	o	o	o
9	o	o	o	o	o	o	o	o	o
10	o	o	o	o	o	o	o	o	o

Nous souhaitons maintenant créer un programme **RemplirZone**, qui va prendre en paramètre un nom de fichier et remplir les différentes zones de l'image.

Nous vous proposons de récupérer l'image :

asterix_trait.png

Nous vous proposons les classes :
RemplirZone.java Point.java

Complétez le méthode `peindreRec` pour remplir les zones de façon récursive.

pour tester vous lancerez la commande : `java RemplirZone asterix_trait.png asterix_couleur.png`

Maintenant vous pouvez tester avec les mers et océans du risk : carte_risk.png

On remarque alors les limites de la récusivité, mais il est possible d'utiliser une File pour pallier au dépassement de capacité de la "Stack".
Nous allons donc écrire une nouvelle méthode `peindreFile`, qui va cette fois-ci utiliser une File.

Travail à Faire

4.3 Ecrire la nouvelle version de cette méthode.
4.4 Adapatez le programme pour que les points de départ du pot de peinture ne soient plus écrits en dur dans le code, mais soient lus dans un fichier data.

Exemple de fichier data pour les mers et océans de notre carte risk.

Nous souhaitons maintenant modifier la couleur de fond de notre image Asterix de référence, en violet (163, 73, 164 ≡ 10701220).
Voici malheureusement le résultat obtenu :

Si on **zoom** sur notre image d'origine, on se rend compte qu'il y a plein de parasites autour du personnage.

Ce nuage parasite est dû aux algorithmes d'anti-alias qui ont permis de construire l'image d'origine.

Nous pouvons nous apercevoir que sur l'espace dédié à notre arrière-plan, qu'il y a des pixels clairs, donc pas très éloignées du blanc.

Nous pouvons calculer la distance entre deux couleurs, par la formule :

$$\sqrt{(rouge1 - rouge2)^2 + (vert1 - vert2)^2 + (bleu1 - bleu2)^2}$$

Deux couleurs identiques ont une distance de 0,
le blanc et le noir ont une distance de 441.67.
En utilisant la bonne distance, nous devrions obtenir l'image suivante:

Travail à Faire

4.5 Récupérez la classe ImageUtil de l'exercice 2, e ajoutez-y la méthode `public static double distance (int coul1, int coul2)` permettant de calculer la distance de deux couleurs.

Adaptez le programme RemplirZone, pour que tous les points ayant une distance inférieure à 90 par rapport à la couleur du point d'origine, soient convertis.

(1/N2) [(N - x).(N - y).fi,j + x.(N - y).fi,j+1 + (N - x).y.fi+1,j + x.y.fi+1,j+1]

Exercice 5 : Fusion d'images

retour haut de page

Nous souhaitons maintenant fusionner deux images. illustration

Voici la commande qu'il faudra lancer : `java Fusionner decor.png asterix_violet.png result.png 10701220 900 625`

param1 : nom du fichier de l'image d'arrière-plan
param2 : nom du fichier de l'image à positionner sur l'arrière-plan. **Cette image doit forcément être plus petite que l'image d'arrière-plan**
param3 : nom du fichier en sortie (au format png)
param4 : couleur dans l'image (param2) à ne pas recopier
param5 : coordonnées x dans l'image param1, du coin supérieur gauche de l'image(param 2)
param6 : coordonnées y dans l'image param1, du coin supérieur gauche de l'image(param 2)

decor.png

Travail à Faire

5 Ecrire le programme répondant à ce besoin.