

Compte rendu TP2 : Programmation générique

Commandeur N. - Verdant B

10/11/20

1 Introduction

Pour compiler les sources, il suffit de faire un make. Chaque réponse fera l'affaire d'une section avec la commande correspondante.

Les fichiers pour tester sont :

- ./patinoire.ppm
- ./patinoire2.ppm
- ./patinoire3.ppm
- ./dessin1.ppm
- ./dessin2.ppm
- ./kowloon.ppm
- ./colors.ppm
- ./lena.pgm

2 Définition d'une classe pour représenter des images arbitraires

Utilisation de la classe générique Image2D pour représenter une image en niveau de gris :

```
$ ./testGrayLevelImage2D
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
5 5 5 5 5 5 5 5
```

3 Introduction des images couleurs

Utilisation de la classe générique Image2D pour représenter une image en couleur.

```
$ ./testColorImage2DBash
255 0 255| 255 0 255| 255 0 255|
255 0 255| 255 0 255| 255 0 255|
255 0 255| 255 0 255| 255 0 255|
```

4 Premier itérateur sur images quelconques

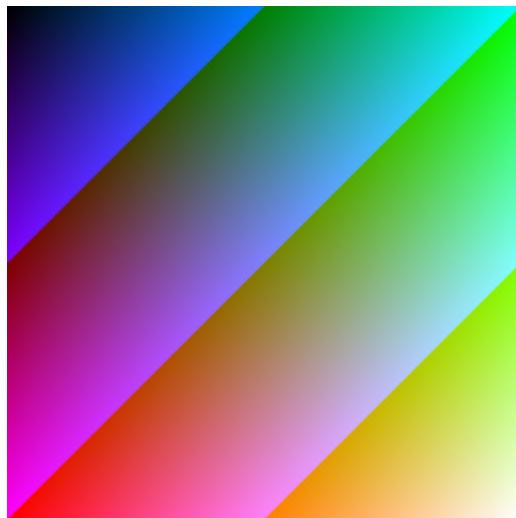
Utilisation de la classe générique Image2D pour représenter une image en couleur,

```
$ ./testColorImage2D
```

5 Un importeur / exporteur PBM générique

On va écrire une classe pour écrire et lire un fichier, en spécialisant pour la couleur et les nuances de gris. On va réutiliser la fonction permettant de créer l'image et l'écrire dans le fichier *colors.ppm*

```
$ ./testColorImage2D  
$ display colors.ppm
```



6 Premier test: on inverse les canaux rouge et bleu

Grâce à nos fonctions de lecture et écriture, on peut commencer à traiter les images. On va lire une image et écrire une nouvelle en changeant la couleur de chaque pixel en inversant le bleu et le rouge. Il faut donner une image en entrée et une image en sortie.

```
$ ./invert-blue-red patinoire.ppm patinoire-inv.ppm  
$ display patinoire-inv.ppm
```



7 On rajoute les accesseurs et un itérateur générique

On va définir des itérateurs sélectifs qui ne voient qu'une composante de la couleur (en lecture et en écriture). Plutôt que de les réécrire à chaque fois, on définit la notion d'accesseur, puis nos nouveaux itérateurs seront paramétrés par un accesseur. Grâce à nos nouveaux accesseur, on va pouvoir sauvegarder les 3 channels de couleurs sous forme de niveau de gris.

```
$ ./save-all-channel patinoire.ppm
$ display patinoire_blue.pgm
$ display patinoire_red.pgm
$ display patinoire_green.pgm
```



(a) Image originale



(b) Image des niveaux de gris du rouge



(c) Image des niveaux de gris du vert

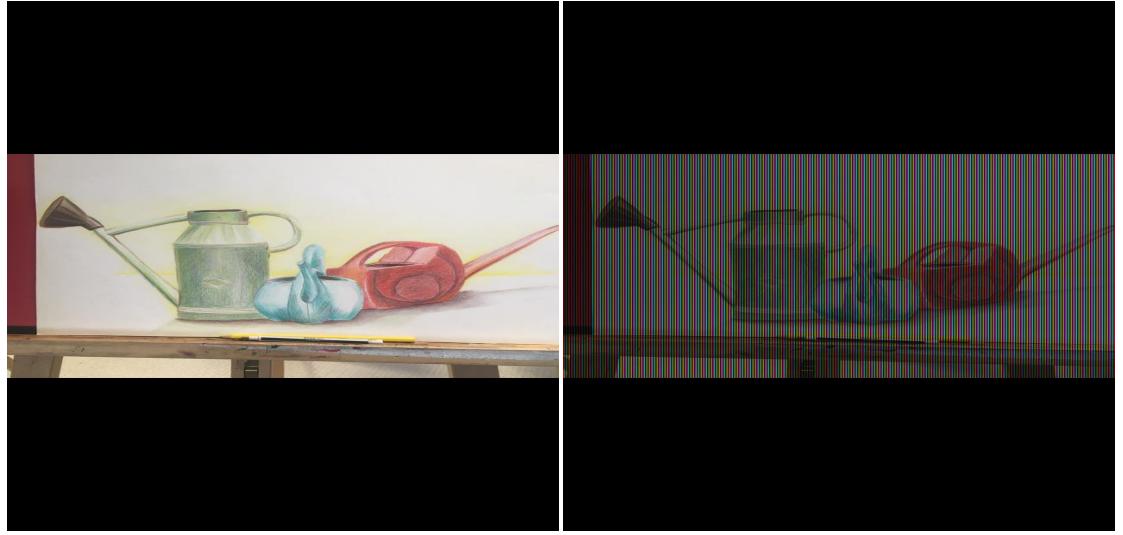


(d) Image des niveaux de gris du bleu

8 Itérateurs génériques non constants

Nous avons fait l'itérateur générique en lecture. Maintenant l'itérateur générique en lecture/écriture (i.e. non const).

```
$ ./save-all-channel dessin1.ppm dessin1_catho.ppm  
$ display dessin1_catho.ppm
```



(a) Image originale

(b) Image comme sur les écrans cathodiques

9 Espace TSV (HSV) et histogramme d'une image couleur

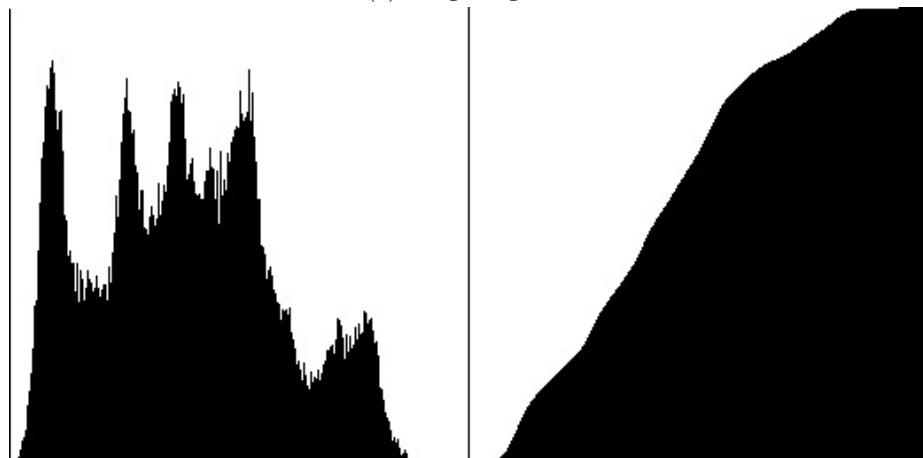
Ici deux programmes, un pour faire des histogrammes sur les images en gris et l'autre pour faire des histogrammes sur les images couleurs.

D'abord celui en gris

```
$ ./histogrammeGrey lena.pgm lenaHisto.pgm  
$ display lenaHisto.pgm
```



(a) Image originale



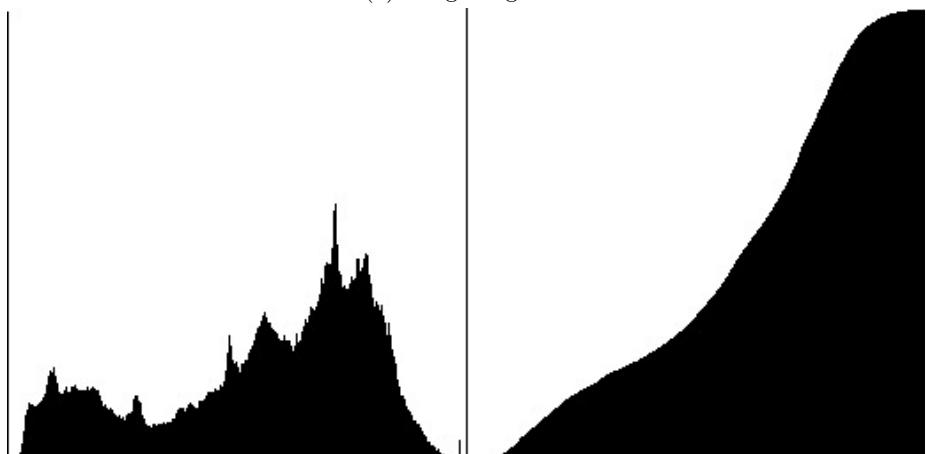
(b) Histogramme des niveaux de gris

Ensuite en couleur

```
$ ./histogrammeColor kowloon.ppm kowloonHisto.pgm  
$ display kowloonHisto.pgm
```



(a) Image originale



(b) Histogramme des niveaux de couleur

10 Egalisation d'image couleur

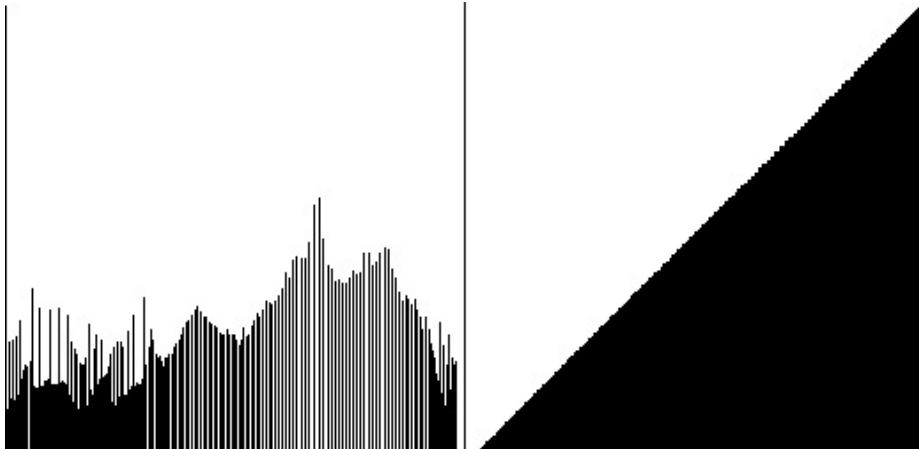
Dès que l'on a un histogramme cumulé d'une image, il est maintenant facile d'égaliser l'image pour la rendre bien balancé. On utilisera cette fonction d'égalisation mais sur les valeurs des couleurs, via nos itérateurs génériques avec accesseur `ColorValueAccessor`.

```
$ ./egaliseur kowloon.ppm
$ display kowloon_histoEq.pgm
$ display kowloon_egalise.pgm
```



(a) Image originale

(b) Image égalisée



(c) Histogramme des niveaux de couleur égalisée

11 Un peu d'imagination

11.1 Inversion des couleurs

Petit programme qui permet d'inverser les valeurs des couleurs d'une image. Il suffit de faire un $ValeurComposante = 255 - ValeurComposante$ sur chaque composante.

```
$ ./invert-color patinoire.ppm patinoireInvert.ppm  
$ display patinoireInvert.ppm
```



(a) Image originale



(b) Image avec couleurs inversées

11.2 Sepia

Petit programme qui permet d'appliquer un filtre sépia sur une image. Il suffit de multiplier chaque composantes par des constantes (cf : Stackoverflow)

```
$ ./sepia patinoire2.ppm patinoire2Sepia.ppm  
$ display patinoire2Sepia.ppm
```



(a) Image originale



(b) Image avec filtre sepia

Conclusion

TP très appréciable à faire, surtout voir le résultat sur des photos perso, mais quelques fois rageant à cause de certaines erreurs très énervantes, notamment la construction de l'histogramme de couleur, où il faut diviser par la valeur max pour éviter un segFault quand on construit le tableau.