Langage C++

- CIR2 -

TP4 : Manipulation des images PGM

Nils BEAUSSÉ (nils.beausse@isen-ouest.yncrea.fr)

Compétences travaillées dans ce TP :

- Encapsulation
- Constructeurs
- Destructeurs
- Traitement d'images
- Fichiers

Consignes :

- Tester dans le main, à chaque fois, chaque méthode développée
- Utiliser le débogage

Principe des images PGM

Le format PGM (Portable GreyMAp) est un format de fichier permettant de stocker des données d'image. Ce type d'image contient :

- Ligne 1 : P2 (magic number indiquant le format d'image)
- Ligne 2 : largeur hauteur (2 valeurs entières indiquant la taille de l'image)
- Ligne 3 : valmax (1 valeur entière indiquant le niveau de gris maximal, souvent 255)
- Ligne 4 et suivantes : les valeurs entières sous forme de matrice représentant l'intensité en niveau de gris de chaque pixel de l'image variant entre 0 (noir) et 255 (blanc). Chaque ligne de l'image est représentée par

une ligne d'entiers (séparés par des espaces) représentant l'intensité en niveaux de gris de chaque pixel de la ligne. En traitement d'image, le coin haut gauche est le point de coordonnées (0,0)

L'extension de ce type d'images est « .pgm »

Exemple des images PGM

Les données ci-dessous

```
P2
24 7
255
0 0 0   0 0 0   0  0  0  0 0  0  0  0  0 0 0 0   0   0   0   0   0 0
0 0 255 0 0 50 50 50 50 0 0 11 11 11 11 0 0 200 0   0   0   200 0 0
0 0 255 0 0 50 0  0  0  0 0 11  0  0  0 0 0 200 200 0   0   200 0 0
0 0 255 0 0 50 50 50 50 0 0 11 11 11  0 0 0 200 0   200 0   200 0 0
0 0 255 0 0 0  0  0  50 0 0 11  0  0  0 0 0 200 0   0   200 200 0 0
0 0 255 0 0 50 50 50 50 0 0 11 11 11 11 0 0 200 0   0   0   200 0 0
0 0 0   0 0 0   0  0  0  0 0  0  0  0  0 0 0 0   0   0   0   0   0 0
```

génère l'affichage suivant :

Pour plus d'information sur ce type d'image, visiter le lien suivant :

https://fr.wikipedia.org/wiki/Portable_pixmap#PGM

Création des images PGM

1) Créer une classe PGM comportant les attributs privés suivants :
    a) largeur : un entier représentant la largeur de l'image
    b) hauteur : un entier représentant la hauteur de l'image

c) valeur_max : un entier représentant la valeur maximale de l'intensité en niveau de gris
d) data : un tableau 2d d'entiers qui contient juste les pixels de l'image sans les trois premières lignes de l'exemple ci-dessus

2) Ecrire les mutateurs qui permettront la modification de ces attributs.
3) Écrire la méthode **void initImage**() permettant d'allouer en mémoire un tableau d'entiers 2d dynamique (int**).
4) Écrire la méthode **void supprImage**() permettant de libérer l'espace alloué par le tableau dynamique **data**
5) Écrire la méthode **void creeImage**(**int** minpix, **int** maxpix) permettant de remplir un tableau 2d (int**) par des valeurs entières aléatoires dans l'interval [minpix, maxpix]. Si ces deux valeurs ne sont pas spécifiées par l'utilisateur, elles sont initialisées à 0 et 255 respectivement. Pour générer les valeurs aléatoires, utiliser la fonction rand (). Exemple d'utilisation : http://www.cplusplus.com/reference/cstdlib/rand/
N'oublier pas d'utiliser la fonction srand (voir tp2) pour ne pas avoir des valeurs aléatoires identiques.
6) Écrire un constructeur par défaut
7) Écrire un deuxième constructeur qui prend comme arguments : hauteur, largeur, valeur minimum des pixels (0 pour le blanc) , valeur maximale des pixels (255 pour le noir). Pour ce faire, n'appeler que les mutateurs
8) Ecrire un destructeur. Pour ce faire, appeler juste une méthode déjà programmée
9) Écrire un constructeur de copie
10) Écrire une méthode **void afficherImage**() permettant d'afficher la matrice des pixels (**data**)

11) Écrire une méthode **void ecrireFichier**(**char*** nom_fichier) permettant de stocker (voir exemple ci-dessus)
    a) P2 dans la ligne 1
    b) La valeur de la largeur suivie d'un espace suivi de la valeur de la hauteur dans la ligne 2
    c) La valeur maximale de niveau de gris dans la ligne 3
    d) La matrice data dans le reste des lignes avec des espaces entre les valeurs d'intensité

    dans le fichier « nom_fichier ».

    Pour manipuler les fichiers en c++, utiliser la classe ofstream : http://www.cplusplus.com/doc/tutorial/files/
    Ce fichier doit avoir l'extension « .pgm ».
    Une fois que le fichier est créé, ouvrez-le dans un visualisateur d'image capable de lire les images pgm afin de visualiser le résultat. (Par exemple Xnview (https://www.xnview.com/fr/) ou IrfanView (à télécharger depuis le lien suivant : https://www.irfanview.com/) ou encore GIMP et Photoshop.)
12) Écrire une méthode permettant de connaitre le nombre d'images PGM créées par l'utilisateur.
    a) Utiliser le débogage pour savoir le nombre d'images PGM créées à la fin du programme
    b) Proposer une solution pour que le nombre de ces images PGM soit exactement 0 à la fin du programme
13) Écrire la méthode **void dessinRect**(**int** x1, **int** y1, **int** x2, **int** y2, **int** val) permettant de dessiner dans **data** un rectangle rempli par les pixels de valeur **val**.

14) Écrire la méthode **void dessinLigne**(**int** x1, **int** x2, **int** line, **int** val) permettant de dessiner dans **data** une ligne horizontale remplie par les pixels de valeur **val**.
15) Écrire la méthode **void dessinCroix**(int x, int y, **int** val) permettant de dessiner dans **data** une croix remplie par les pixels de valeur **val**. On se restreint à une croix de taille : hauteur=1 et largeur=1

Opérations sur les images PGM

Les méthodes développées dans cette partie seront appliquées sur les images suivantes : image1 et image2

1) Compléter le contenu de la méthode ci-dessous qui met à jour les valeurs des attributs de la classe PGM en se basant sur le fichier « nom_fichier.pgm ».
   N'oubliez pas d'ajouter les en-têtes nécessaires.

```
void Pgm::lectureFichier(char* nom_fichier) {
   ifstream monfichier;
   string ligne;
   stringstream s;
   monfichier.open(nom_fichier);
   if (monfichier.is_open()){
       getline(monfichier,ligne);
       s << monfichier.rdbuf();
       s >> largeur >> hauteur;
       // à compléter
}
   monfichier.close();
}
```

2) Créez la méthode **void seuil**(**int** seuil) qui permet d'appliquer un seuillage binaire à l'image PGM et enregistrez le résultat dans un nouveau fichier

« nom_fichier .pgm ». L'application du seuillage sur une image à niveaux de gris est définie comme suit :

$$I_{dst}(i,j) = \begin{cases} 255 \; si \; I_{src}(i,j) > seuil \\ 0 \; sinon \end{cases}$$

3) Appliquer les méthodes **dessinRect**, **dessinLigne**, **dessinCroix** sur l'image « lena.pgm » et stocker les résultats dans 3 fichiers distincts
4) Créer une méthode void **flou** qui floute l'image en fonction d'une largeur de pixels voisins donnée en argument.
5) Créer une méthode **void filtrerImage**(int k) qui permet d'appliquer un filtre médian de taille k*k (https://fr.wikipedia.org/wiki/Filtre_m%C3%A9dian) sur l'image bruitée (« balloons_noisy.ascii ») et enregistre le résultat dans un nouveau fichier « nom_fichier.pgm ».
   L'algorithme nécessite de trier un tableau, vous pouvez implémenter le tri en choisissant l'un des algorithmes de tri vu en cours d'algo.

Extensions aux images PPM

Créer une nouvelle classe PPM (https://fr.wikipedia.org/wiki/Portable_pixmap#PPM) qui reprend le même principe que PGM avec, cette fois-ci, 3 valeurs pour chaque pixel représentant l'intensité de chaque composante couleur R(Red), G(Green), B(Blue).

Ajoutez aux méthodes de la classe un choix de couleur et testez-les.

Adaptez et testez les méthodes de valeurs médianes et de flou aux images avec couleurs.