

Programmation en C et structures de données

guillaume.revy@univ-perp.fr

CC1, Jeudi 6/11/2024 de 14h à 17h

Ce programme est à réaliser seul, sous environnement GNU/Linux. Il doit être rendu sur Moodle, sous forme d'un unique fichier <nom_prenom.c>. Seuls les documents disponibles sur Moodle sont autorisés. Tout autre document (cours, exercices, ...), ou autre matériel (téléphone portable, ...) est interdit.

Conseils :

- Il est à noter que la qualité primera sur la quantité : un programme court, clair, bien conçu et qui compile sera mieux noté qu'un gros programme mal conçu, incompréhensible, voire qui ne compile pas. Utilisez des noms de variables explicites, indentez votre programme, et n'hésitez pas à le commenter.
- Lisez tout le sujet avant de commencer.
- Veuillez respecter les noms de fonctions et structures proposées dans le sujet.

Le but du projet est de travailler sur le traitement d'images. Le programme permettra de lire un fichier image, de lui appliquer des transformations ou des filtres (niveau de gris, flou, ...), et finalement de sauvegarder l'image résultat dans un autre fichier.

Dans le cadre de ce projet, nous allons manipuler des images au format *portable pixmap* (PPM), qui permet de représenter des images en couleur. Plus particulièrlement, dans ce format, une image est représentée par un fichier ASCII, dans lequel aucune ligne ne doit dépasser 70 caractères, et dans lequel les lignes commençant par le caractère # sont ignorées (commentaires). Toutes les valeurs (code image, pixel, ...) sont codées en ASCII et séparées par un caractère d'espacement (espace, tabulation, ou nouvelle ligne). L'image est codée dans le fichier ligne par ligne, de haut en bas et de gauche à droite. Une image de test (`lena.ppm`) est jointe à ce sujet.

Le contenu du fichier est le suivant :

- un nombre magique (P3, pour PPM ASCII),
- la largeur et la hauteur de l'image,
- la valeur maximale utilisée pour coder les couleurs (≤ 65536 , représentable sur 16 bits),
- les données de l'image : un pixel est représenté par trois valeurs successives représentant les composantes rouge, verte et bleue du pixel (RGB).

1. Définition des pixels et des images

Nous allons dans un premier temps définir la structure qui contiendra les données de l'image, à savoir :

- la largeur et la hauteur de l'image,
- la valeur maximale utilisée pour coder les couleurs (≤ 65536),
- un tableau 2D de pixels.

Un pixel est représenté par une structure contenant les composantes R, G, et B sous forme de trois entier 16 bits. Et les lignes du tableau de pixels sont indiquées de haut en bas et les colonnes, de gauche à droite.

Pour le tableau 2D de pixels, vous utiliserez :

- soit des pointeurs et de l'allocation dynamique,
- soit un tableau 2D statique de taille suffisamment grande (i.e. 512×512 , correspondant à la taille de l'image test).

Dans la suite de l'énoncé, nous considérerons la version SANS allocation dynamique.

- ▶ 1. Écrire la définition deux structures précédentes, nommées `pixel` et `image_ppm`.
- ▶ 2. Écrire une fonction `charger_ppm` qui charge une image PPM à partir d'un fichier dont le nom est passé en paramètre. *Nous considérons ici que le fichier d'entrée ne contient aucune ligne de commentaire.*
- ▶ 3. Écrire une fonction `copier_ppm` qui copier une image PPM dans une autre.
- ▶ 4. Écrire une fonction `sauvegarder_ppm` qui sauvegarde une image PPM dans un fichier dont le nom est passé en paramètre.

2. Une première transformation d'images

Nous allons maintenant définir une première transformation applicable sur une image.

Remarques : dans la suite de l'énoncé, nous noterons $P_{\text{src}(i,j)}$ le pixel aux coordonnées (i, j) de l'image source et $P_{\text{dest}(i,j)}$ celui aux coordonnées (i, j) de l'image résultat. Les composantes R, G, B du pixel $P_{\text{src}(i,j)}$ seront notées, respectivement :

$$R(P_{\text{src}(i,j)}), \quad G(P_{\text{src}(i,j)}), \quad \text{et} \quad B(P_{\text{src}(i,j)}).$$

Lors de la transformation d'une image, l'image source devra rester inchangée.

Cette première transformation consiste à appliquer à une image source une symétrie suivant l'axe des abscisses et/ou des ordonnées. Par exemple, si l'on applique une symétrie suivant l'axe des ordonnées à une image de largeur k , le pixel (i, j) de cette image sera en coordonnées $(i, k - 1 - j)$ de l'image destination :

$$P_{\text{dest}(i,k-1-j)} = P_{\text{src}(i,j)}.$$

- 1. Écrire une fonction `renverse_ppm`, qui prend en paramètre une image et l'axe suivant lequel l'image doit être renversée, et retourne une copie de l'image après lui avoir appliqué la symétrie.

Remarque : l'axe suivant lequel l'image doit être renversée pourra être représenté avec un entier 8 bits.

3. Modification de la luminosité et du contraste d'une image

Nous allons maintenant voir comment modifier la luminosité et le contraste d'une image.

Luminosité. Pour modifier la luminosité d'une image, il suffit de multiplier chaque pixel, ou une ou plusieurs de leurs composantes, par un scalaire α .

- 1. Écrire une fonction `luminosite_ppm` qui prend en paramètre une image, le scalaire α , et la ou les composantes à modifier (sous forme de caractère 'r', 'g', 'b', ou 'a' pour toutes les composantes), puis qui retourne une copie de cette image après avoir modifier sa luminosité en fonction de α .

Contraste. Pour modifier la contraste d'une image, il suffit d'additionner un entier naturel à chaque pixel, ou à une ou plusieurs de leurs composantes.

- 2. Écrire une fonction `contraste_ppm` qui prend en paramètre une image, un entier naturel, et la ou les composantes à modifier (sous forme de caractère 'r', 'g', 'b', ou 'a' pour toutes les composantes), puis qui retourne une copie de cette image après avoir modifier son contraste.

Dans ces deux cas, si la valeur d'un pixel ou de chaque composante est négative, elle est fixée à 0, et si elle dépasse la valeur maximale de l'image, elle est fixée à la valeur maximale.

4. Application de filtres sur les images

Nous allons maintenant définir ce qu'est un filtre sur une image. Un filtre sera défini par une matrice F carrée de taille $n \times n$ (on considérera ici $n \in \{3, 5\}$) et deux paramètres entiers a et b . La valeur de chaque composante du pixel $P_{\text{dest}(i,j)}$ est définie de la manière suivante :

$$P_{\text{dest}(i,j)} = \max \left(\frac{\sum_{k_1=-k}^k \sum_{k_2=-k}^k P_{\text{src}(k+i,k+j)} \cdot F_{k+k_1,k+k_2}}{a} + b, 0 \right), \quad \text{avec } k = \lfloor n/2 \rfloor$$

Nous appliquerons un filtre sur une image, pixel par pixel, composante par composante.

Vous utiliserez ici une matrice de taille maximum 5×5 , et des paramètres a et b sous forme d'entiers 32 bits.

- 1. Définir la structure `filtre` précédente.
- 2. Écrire la fonction `fuzzy_filter`, qui prend en paramètre la valeur n , et retourne une instance d'un filtre permettant d'appliquer un flou à une image, définie de la manière suivante (pour $n = 3$ et $n = 5$).

$$F = \begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}$$

avec $n = 3, a = 9$, et $b = 0$

$$F = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

avec $n = 5, a = 25$, et $b = 0$

- 3. Écrire les fonctions `prewitt_x_filter` et `prewitt_y_filter`, qui prennent en paramètre une image et la valeur n , et retourne une instance d'un filtre de Prewitt en X et en Y, respectivement, définis de la manière suivante (pour $n = 3$).

$$F = \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix}$$

avec $n = 3, a = 6$, et $b = \lceil \text{valeur maximale}/2 \rceil$

$$F = \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

avec $n = 3, a = 6$, et $b = \lceil \text{valeur maximale}/2 \rceil$

- 4. Écrire les fonctions `sobel_x_filter` et `sobel_y_filter`, qui prennent en paramètre une image et la valeur n , et retourne une instance d'un filtre de Sobel en X et en Y, respectivement, définis de la manière suivante (pour $n = 3$).

$$F = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}$$

avec $n = 3, a = 8$, et $b = \lceil \text{valeur maximale}/2 \rceil$

$$F = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

avec $n = 3, a = 8$, et $b = \lceil \text{valeur maximale}/2 \rceil$

- 5. Écrire la fonction `discret_laplace_filter`, qui prend en paramètre une image, et retourne une instance d'un filtre Laplacien discret défini de la manière suivante (pour $n = 3$).

$$F = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

avec $n = 3, a = 1$, et $b = \lceil \text{valeur maximale}/2 \rceil$

- 6. Écrire une fonction `apply_filter_pixmap`, qui prend en paramètre une image et un filtre, copie l'image, applique le filtre à la copie de l'image, et la retourne.