Premier devoir noté 2016 – Traitement d'images numériques

Laurent Bindschaedler Jean-Cédric Chappelier

du mercredi 16 mars 12:00 au lundi 04 avril 23:59

Ce devoir noté consiste à écrire un programme permettant de créer une image numérique simplifiée, l'afficher (en mode texte), l'écrire et la lire depuis le disque ainsi que de lui appliquer un filtrage par convolution.

Vous écrirez tout votre code dans un seul fichier nommé `muimp.c` (pour « micro (μ) **im**age **p**rocessor »).

I. Création d'une image

Une image numérique est définie comme un tableau à 2 dimensions de taille `hauteur * largeur` contenant des pixels.

Définissez un *type* `Pixel` correspondant à un pixel, lesquels ont une valeur réelle (`double`). Utilisez ensuite ce type pour créer une *structure de données* `Image` contenant un tableau à double entrée de `Pixel`. Ce tableau sera alloué statiquement aux dimensions définies dans des constantes `MAX_IMAGE_HEIGHT` et `MAX_IMAGE_WIDTH`, les deux de valeur 100.

Attention: `MAX_IMAGE_HEIGHT` et `MAX_IMAGE_WIDTH` définissent les dimensions *maximales*, mais pas forcément les dimensions exactes de l'image ! Vous devez donc ajouter deux attributs (`height` et `width`) à chaque image, correspondant aux dimensions exactes.

Note : l'allocation statique est une mauvaise chose en général, typiquement dans ce cas (objet de taille inconnue). Mais nous n'avons pas encore abordé en cours l'allocation dynamique, qui viendra plus tard. N'ayez donc pas l'impression que, parce que nous vous l'imposons dans ce premier devoir, l'allocation statique soit une bonne chose. Non ! Elle est effectivement à proscrire en général (sauf cas particuliers).

Définissez ensuite une fonction `diamond` qui retourne une image remplie avec un losange au centre de l'image. La valeur des pixels à l'intérieur du losange est

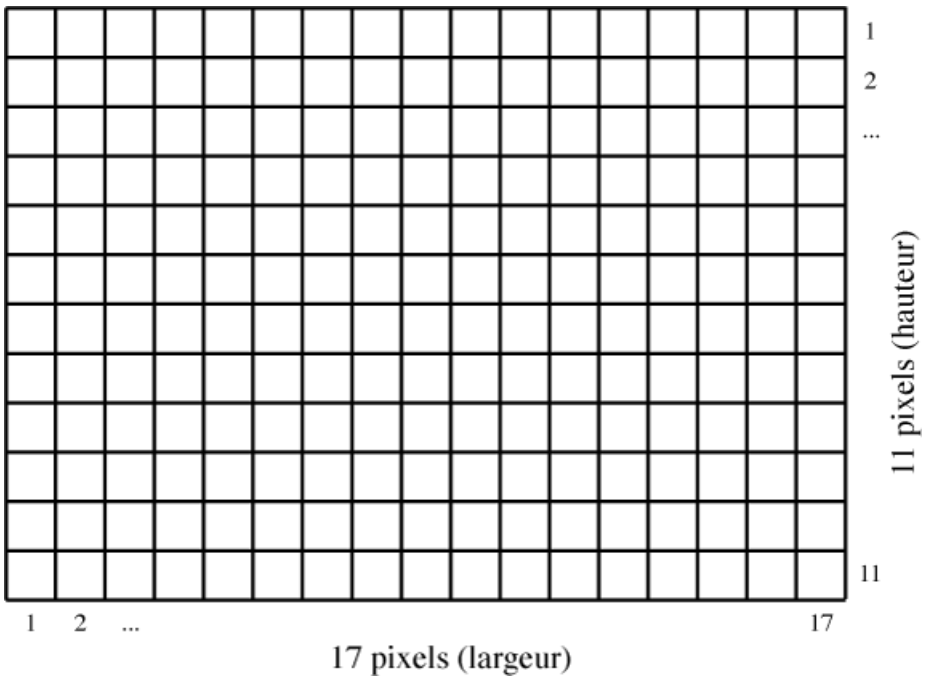

Figure 1: Une image est un tableau de pixels.

1.0 et celle des pixels à l'extérieur du losange de 0.0 (cf. exemple dans la section suivante).

Les dimensions de l'image ainsi que la taille de la diagonale du losange (on supposera les deux diagonales égales : c'est en fait un carré, pivoté de 45 degrés) doivent être passées en argument à la fonction `diamond`.

Pour créer le losange, déterminez pour chaque ligne y_p de l'image les deux pixels de chaque bords du losange. Ces points $P_1(x_1, y_p)$ et $P_2(x_2, y_p)$ vérifient les équations suivantes :

$$x_1 = x_c - (y_p - y_c + D/2)$$

$$x_2 = x_c + (y_p - y_c + D/2)$$

où D est la diagonale et $C(x_c, y_c)$ le centre de l'image.

Consigne : Utiliser la symétrie horizontale du losange pour ne parcourir que la moitié des lignes de l'image.

Note : pour ne pas avoir d'ambiguïté sur le centre de l'image, on ne considérera que des images de taille impaire. Pour forcer cela, utiliser le code suivant :

```
// Make sure width and height are odd
width  |= 1;
height |= 1;
```

Dans le `main()`, appliquer la fonction losange à l'image précédemment définie. Demandez à l'utilisateur de choisir la taille de la diagonale ainsi que les dimensions de l'image. N'oubliez pas d'effectuer un *contrôle strict* des entrées et traiter les cas particuliers ! Votre programme *devra* être robuste.

Et pensez à modulariser votre code… (ça devrait aller de soit, je ne vous le rappellerai plus !)

II. Affichage de l'image

Dans une vraie image numérique, la valeur de chaque pixel représente une couleur. Pour simplifier, nous utiliserons ici des symboles prédéfinis en fonction de la valeur du pixel.

Nous vous demandons d'étendre votre fichier `muimp.c` précédent en définissant une fonction `display` qui prend en argument un flot (`FILE*`) et une image numérique telle que définie précédemment et qui, pour chaque pixel de l'image, affiche dans le flot :

- « `.` » si sa valeur est 0,
- «`+`» si sa valeur est 1,
- «`*`» sinon.

Dans le `main()`, utiliser la fonction `display` pour afficher sur la sortie standard l'objet image créé précédemment. Par exemple, pour une image de `11 * 17` et une diagonale de `10`, vous obtiendrez le résultat suivant :

```
. . . . . . . . + . . . . . . . .
. . . . . . . . + + + . . . . . .
. . . . . . + + + + + . . . . . .
. . . . . + + + + + + + . . . . .
. . . . + + + + + + + + + . . . .
. . . + + + + + + + + + + + . . .
. . . . + + + + + + + + + . . . .
. . . . . + + + + + + + . . . . .
. . . . . . + + + + + . . . . . .
. . . . . . . + + + . . . . . . .
. . . . . . . . + . . . . . . . .
```

III. Ecriture et lecture d'images depuis un fichier

Ecrivez une fonction `write_to_file` qui prend un nom fichier et une image en paramètre et qui écrit dans le fichier : * la largeur de l'image, sur une ligne ; * la hauteur de l'image, sur une ligne ; * le dessin de l'image comme ci-dessus.

En cas d'erreur (ouverture du fichier), la fonction `write_to_file` retournera un entier non nul. Elle retournera `0` en cas de succès.

Ecrivez également une fonction `read_from_file` qui prend un nom de fichier (correspondant au format précédent), lit l'image depuis le fichier et la retourne. En cas d'erreur (soit on n'arrive pas à lire les tailles, soit elles sont trop grandes, soit on n'arrive pas à lire toute l'image, etc.), la fonction retournera une image dont la largeur ou la hauteur sont à `0`.

Dans le `main()`, demandez à l'utilisateur de fournir un nom de fichier, utilisez la fonction `write_to_file` pour écrire l'objet image créé précédemment dans ce fichier.
Ensuite, ouvrez ce même fichier en lecture et utilisez la fonction `read_from_file` pour lire l'image à nouveau et l'afficher. Elle devrait être identique à l'image affichée ci-dessus.

Là aussi, n'oubliez pas d'effectuer un *contrôle strict* de toutes les informations et situations. Votre programme *doit* toujours être robuste…

IV. Filtrage d'image

Une méthode très utilisée en traitement d'images numériques est l'utilisation du « filtrage par convolution ». Cela consiste à calculer, pour chaque pixel de l'image, la somme des produits de chaque point du « masque de convolution » avec le point de l'image correspondant :

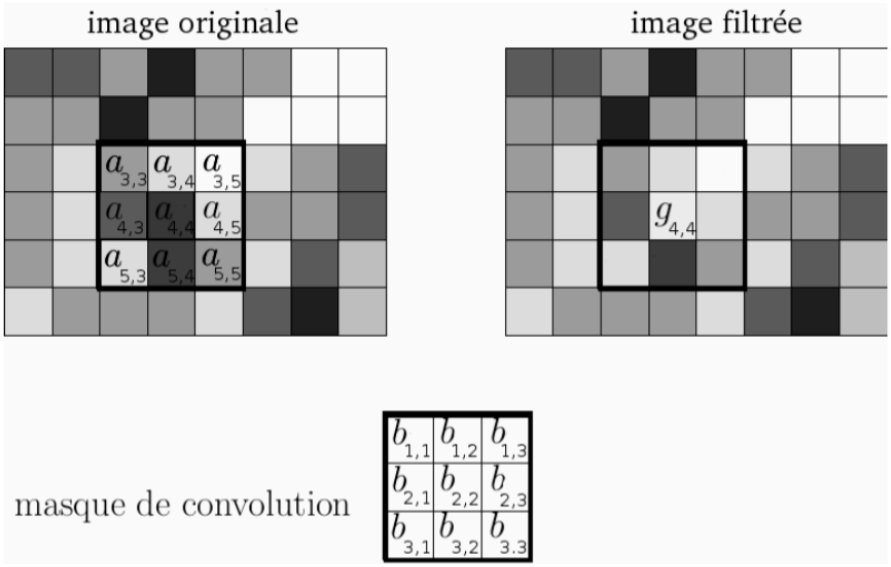

Figure 2: Exemple de filtrage sur image

Si l'on note $g_{i,j}$ la valeur du pixel (i, j) de l'image filtrée, $a_{i,j}$ celle du pixel de l'image d'origine, et b le masque (de taille N), alors :

$$g_{i,j} = \sum_{k=1}^{N} \sum_{l=1}^{N} a_{i+\lfloor N/2 \rfloor - k + 1, j + \lfloor N/2 \rfloor - l + 1} \cdot b_{k,l}$$

(la formule est donnée ici en « indices mathématiques », compris entre 1 et la taille correspondante ; si l'on indexe entre 0 et taille moins 1, on aurait :

$$g_{i,j} = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} a_{i+\lfloor N/2 \rfloor - k, j + \lfloor N/2 \rfloor - l} \cdot b_{k,l}$$

).

On vous demande d'étendre à nouveau votre fichier `muimp.c` pour ajouter une fonction de filtrage d'image.

Pour commencer, définissez dans le `main()` un masque de filtrage tel que celui-ci :

```
-2.0   -2.0   -2.0
 0.0    0.0    0.0
 2.0    2.0    2.0
```

Pour simplifier, utilisez le type `Image` pour représenter un masque de filtrage.

Définissez ensuite une fonction `filter` prenant en paramètre une image et un masque et qui retourne une nouvelle image, résultat du filtrage correspondant à l'application de la formule ci-dessus.

Pour les points situés sur le bord de l'image, la formule précédente utilise des valeurs indéfinies (points hors de l'image).
Dans ce cas, nous considérons l'image comme périodique : la valeur du pixel suivant le dernier pixel de chaque ligne et colonne est la valeur du premier pixel de la même ligne/colonne (utilisez l'opérateur « modulo »).

Si l'on applique ce filtre à l'image précédente, on devrait obtenir :

```
. . . . . * * * * * . . . . . .
. . . . . * * * * * * * . . . . .
. . . . * * * * * . * * * * * . . . .
. . . * * * * * . . . * * * * * . . .
. . * * * * * . . . . . * * * * * . .
. . . . . . . . . . . . . . . . . .
. . * * * * * . . . . . * * * * * . .
. . . * * * * * . . . * * * * * . . .
. . . . * * * * * . * * * * * . . . .
. . . . . * * * * * * * . . . . . .
. . . . . . * * * * * . . . . . . .
```

Terminez en complétant le `main()` pour afficher l'image ci-dessus en appelant la fonction de filtrage sur l'image de départ.

V. Rendu

Rendez votre programme `muimp.c` via le site Moodle du cours (http://moodle.epfl.ch/course/view.php?id=6
; en bas) avant le **lundi 04 avril à 23:59**.