L'objectif

Le but de ce TD est de se familiariser avec :
- l'allocation dynamique de mémoire et l'arithmétique des pointeurs.
- lecture des fichiers.

Dans ce TD, vous devez écrire des fonctions qui charge une image à partir d'un fichier BMP [1]. Ecrire des fonctions qui effectuent quelques technique de traitement d'image (transformé une image en échelle de gris, effectuer des produits de convolution afin d'appliquer des filtres, ...).

Lisez l'intégralité du document avant d'écrire le code.

Généralités, pixels et couleurs

Les couleurs sont une sorte de données, mais comment peuvent-elles être représentées dans les ordinateurs ? Il y a tant de façons de représenter la couleur, la plus célèbre est la représentation RGBA (RGBA signifie Red, Green, Bleu et Alpha). La plupart des couleurs peuvent être représentées en utilisant un mélange des 3 couleurs fondamentales (le rouge, vert et bleu). L'absence de ces 3 couleurs signifie le noir et leur présence tous ensemble représente la couleur blanche. L'alpha représente la transparence et ce n'est pas très pertinent dans ce que nous allons faire. Généralement, 8 bits sont plus que suffisants pour stocker chacune de ces 3 couleurs et la transparence. Ainsi, chaque canal de couleur est représenté par 1 octet en mémoire (1 octet pour le rouge, le vert, le bleu et l'alpha) ce qui fait heureusement jusqu'à 4 octets. L'ordre des octets rouge, vert, bleu et alpha varie en fonction du format. Les formats les plus courants sont RGBA et BGRA. Dans le premier format, le rouge est stocké en premier, suivi du vert, du bleu et de l'alpha. Dans le second, le bleu est stocké en premier, suivi du vert, du rouge et de l'alpha.
Il existe deux types d'images, les images vectorielles et les images matricielles. Dans ce dernier, (qui est celui que nous allons traiter) les informations de couleur sont stockées par pixel [2] qui sont les plus petits éléments adressables dans ce type d'images. Une image contient donc $largeur * hauteur$ pixels.
Pour la sutie du TD, si on a une matrice M de pixels de taille H par L, ou H l'hauteur et L la largeur de l'image alors `pixels[y][x]` nous donne le pixel à la ligne `y` et la colonne `x`.

Le format de fichier BMP

Le format BMP est un format de fichier d'image graphique de type raster [3], utilisé pour stocker des images numériques. Le format BMP est connu pour sa simplicité car aucune compression n'est effectuée [4].
Un fichier BMP contient au moins 3 sections :

Bitmap File Header

Cette entête, qui a une taille globale de 14 octets, existe au début de chaque fichier BMP. Elle contient des métadonnées sur le fichier en général. Les champs de cette entête sont dans l'ordre :
- **Signature :** Utilisé pour vérifier que le format de fichier est BMP. Il occupe deux octets en mémoire. le premier octet doit être égal au caractère ascii `B` et le second doit être égal au caractère ascii `M`.
- **File size :** Occupe 4 octets. Il est utilisé pour indiquer la taille du fichier en octets.
- **Reserved :** Cela occupe au total 4 octets et est réservé à l'application qui crée l'image. Cela ne sera pas très intéressant pour nous.
- **Offset to pixels :** C'est un champ très important, qui indique combien d'octets nous devons sauter à partir du début du fichier pour trouver le tableau de pixels qui contient des informations sur la couleur de chaque pixel (plus à ce sujet plus tard)

1. Bitmap image file
2. abréviation de picture elements
3. Cela signifie qu'il stocke des pixels contrairement aux images vectorielles
4. en fait, certaines techniques de compression peuvent être utilisées mais ce n'est pas très courant

DIB Header

Cet en-tête se trouve juste après le *Bitmap File Header*. Cette section est importante car elle stocke des informations détaillées sur l'image bitmap et définit le format de pixel. Parmi les nombreux champs que cet en-tête définit, nous ne serons intéressés que par quelques-uns d'entre eux : taille de l'en-tête, largeur, hauteur, nombre de plans de couleur, les bits par pixel et masque de bits du rouge, bleu, vert et alpha. Les premiers champs que cet en-tête contient sont dans l'ordre :

Champ	**Taille**	**Type**	**Description**
`Header Size`	4o	int	la taille de cet section
`Image Width`	4o	int	la largeur du bitmap en pixels (de l'image)
`Image Height`	4o	int	la hauteur du bitmap en pixels (de l'image)
`Planes`	2o	unsigned short	le nombre de plans de couleur (doit être 1)
`Bits per pixel (BPP)`	2o	unsigned short	le nombre de bits par pixel, qui est la profondeur de couleur de l'image. Les valeurs typiques sont 1, 4, 8, 16, 24 et 32.
`Compression`	4o	int	la méthode de compression utilisée (Pas important dans notre cas).
`X pixels per meter`	4o	int	la résolution horizontale de l'image (pixel par mètre). (Pas important dans notre cas)
`Y pixels per meter`	4o	int	la résolution verticale de l'image (pixel par mètre). (Pas important dans notre cas)
`Colors in color table`	4o	int	le nombre de couleurs dans la palette de couleurs. (Pas important dans notre cas)
`Important color count`	4o	int	le nombre de couleurs importantes utilisées (généralement ignoré).
`Red channel bitmask`	4o	unsigned int	Masque binaire de canal rouge. En utilisant un `ET` binaire avec les données de pixel ca va donner la valeur de couleur rouge du pixel
`Green channel bitmask`	4o	unsigned int	Masque binaire de canal vert. En utilisant un `ET` binaire avec les données de pixel ca va donner la valeur de couleur vert du pixel
`Blue channel bitmask`	4o	unsigned int	Masque binaire de canal bleu. En utilisant un `ET` binaire avec les données de pixel ca va donner la valeur de couleur bleu du pixel
`Alpha channel bitmask`	4o	unsigned int	Masque binaire de canal alpha, En utilisant un `ET` binaire avec les données de pixel ca va donner la valeur de l'alpha du pixel

Les champs restants dans l'en-tête DIB ne sont pas pertinents dans notre cas, donc on va les ignorers.

Pixel Array

Il s'agit d'un tableau contenant les données de chaque pixel de l'image. Les données de pixels sont stockées telles qu'elles apparaissent dans l'image en allant de gauche à droite et de haut en bas. Ce qui signifie que le premier pixel du tableau correspond au pixel en haut à gauche de l'image.
Chaque ligne du tableau de pixels est complétée à un multiple de 4 octets de taille. Heureusement pour nous, nous ne traiterons que des données de pixels de 4 octets, donc nous n'avons de toute façon pas à gérer ce padding.

Voici un tableau qui représente la façon dont le tableau de pixels est stocké dans le fichier bmp. `h` représente la hauteur et `w` représente la largeur de l'image.

Données d'image Tableau de pixels [y][x]					
pixel[h-1][0]	pixel[h-1][1]	pixel[h-1][2]	...	pixel[h-1][w-1]	Padding
pixel[h-2][0]	pixel[h-2][1]	pixel[h-2][2]	...	pixel[h-2][w-1]	Padding
pixel[h-3][0]	pixel[h-3][1]	pixel[h-3][2]	...	pixel[h-3][w-3]	Padding
... ...					
pixel[2][0]	pixel[2][1]	pixel[2][2]	...	pixel[2][w-1]	Padding
pixel[1][0]	pixel[1][1]	pixel[1][2]	...	pixel[1][w-1]	Padding
pixel[0][0]	pixel[0][1]	pixel[0][2]	...	pixel[0][w-3]	Padding

Produit de convolution sur les images

L'image est une forme d'un signal discret bidimensionnel. Par conséquent, les filtres peuvent être appliqués en utilisant des produits de convolution. Faire des produits de convolution sur des images est assez simple. Vous

prenez un noyau (également appelé matrice de convolution) et le faites glisser à travers l'image en multipliant chaque cellule de la matrice par le pixel correspondant, puis il suffit d'additionner tous les valeurs obtenus. Et enfin mettre le pixel au centre de la matrice de convolution à la somme que nous avons obtenue.
Si cette explication vous semble horrible, je vous recommande de consulter cet article. Ce n'est pas si long et ca vous offre une manière interactive de visualiser les produits de convolution sur les images.

Travail demandé :

Le travail requis est divisé en parties. A la fin de chaque partie vous pourrez vérifier votre travail en exécutant les tests. Clonez ce repistory Github avant de passer aux questions.

Partie 1 : Quelques fonctions utilitaires : Mémoire dynamique, matrices

Dans cette partie, vous devrez implémenter quelques fonctions utilitaires afin de passer à la partie suivante.

1- Dans le fichier `matrix.c` implémentez une fonction `mat_init(unsigned N, unsigned M, unsigned unit_size)` qui alloue de la mémoire pour une matrice N par M d'un élément de taille `unit_size` octets. Implémenter la fonction `mat_free(void** mat)` qui libère la mémoire allouée à la matrice. Si vous êtes bloqué, vous pouvez utiliser le dossier nommé matrix qui contient `matrix.h` et `matrix.c` les copier dans votre répertoire source principal.

```
/**
* Allocates memory for N by M matrix that contain elements of unit_size bytes
*/
void** mat_init(unsigned N, unsigned M, unsigned unit_size);

/**
* Frees the memory allocated by mat_init
*/
void mat_free(void** mat);
```

Toutes les fonctions qui suit dans cette PARTIE doivent être définies avec le mot-clé `static` et elles doivent être définies en haut du fichier `image.c` juste après les includes.
2- Implémentez une fonction appelée `clz(bits)` [5] qui prend un entier positif comme argument et compte le nombre de bits mis à zéro en allant de gauche à droite jusqu'à ce qu'il atteigne un 1 ou la fin. (Il compte le nombre de zéros avant la première occurrence d'un)
Exemples :

```
Pour 00000000 00000000 00000000 00010000 clz doit retourner : 27
Pour 00000001 00000000 00000000 00010000 clz doit retourner : 7
Pour 00000000 00001000 11111111 00010000 clz doit retourner : 12
Pour 00000000 00000001 11111111 00010000 clz doit retourner : 15
```

C'est une question du premier TD. Si vous avez implémenteé cette fonction avec succès, vous pouvez passer à la question suivante. Si vous ne l'avez pas implémenté, c'est probablement le moment de le faire. Si vous êtes bloqué, pas de soucis, vous pouvez toujours passer à la question suivante juste avant cela, vous devez décommenter cette ligne `#include "clz/clz.h"` dans `image.c` comme indiqué dans les commentaires afin que vous puissiez utiliser l'implémentation que j'ai fourni ;).

3- Toujours en haut du fichier `image.c` après les includes, écrivez une fonction statique qui, étant donné un nombre impair `N`, il recherche les coordonnées des cellules voisines de la cellule centrale dans la matrice NxN.
Exemples : Pour N = 3, la fonction doit retourner le tableau : { `(-1,-1) (-1,0) (-1,1) (0,-1) (0,0) (0,1) (1,-1) (1,0) (1,1)`}
En effet :

(-1, -1)	(-1, 0)	(-1, 1)
(0, -1)	(0, 0)	(0, 1)
(1, -1)	(1, 0)	(1, 1)

Pour N = 5, la fonction doit retourner le tableau : { `(-2,-2) (-2,-1) (-2,0) (-2,1) (-2,2) (-1,-2) (-1,-1) (-1,0) (-1,1) (-1,2) (0,-2) (0,-1) (0,0) (0,1) (0,2) (1,-2) (1,-1) (1,0) (1,1) (1,2) (2,-2) (2,-1) (2,0) (2,1) (2,2)`}
En effet :

5. count leading zeros

(-2, -2)	(-2, -1)	(-2, 0)	(-2, 1)	(-2, 2)
(-1, -2)	(-1, -1)	(-1, 0)	(-1, 1)	(-1, 2)
(0, -2)	(0, -1)	(0, 0)	(0, 1)	(0, 2)
(1, -2)	(1, -1)	(1, 0)	(1, 1)	(1, 2)
(2, -2)	(2, -1)	(2, 0)	(2, 1)	(2, 2)

La fonction doit avoir le profil suivant :

```
/**
* Given an impair kernel_size calc_nehgbours will return the (y, x) coordinates
    relative
* to the central cell. This function returns the coordinates in a linear array where
* the first element is the coordiante of the top left cell going from left to right
    and from
* top to bottom
*/
static vec2* calc_neighbours(unsigned kernel_size)
```

Si vous êtes bloqué, vous pouvez décommenter l'include du fichier `neighbours/neighbours.h` afin de pouvoir utiliser la fonction plus tard.

4- Il est temps de consolider et de tester votre travail. Compilez ce que vous avez fait et exécutez les tests en utilisant respectivement les commandes `make` et `make run`. Si vous avez réussi le premier ensemble de tests (`Test Set 1`), félicitations, vous pouvez passer à la partie suivante.

Partie 2 : Lecture des fichiers

Dans le fichier `bmp_loader.c`, vous devrez implémenter les fonctions suivantes.

1- Implémentez la fonction `bmp_load(bmp* bmp, const char* filename)` qui prend un pointeur vers une structure bmp et un nom de fichier. La fonction doit ouvrir le fichier appelé `filename`, puis appelle `bmp_load_header`, vérifier qu'il s'agit d'un fichier BMP en vérifiant la signature, puis elle appelle `bmp_load_dib_header` et `bmp_load_pixels` avant de fermer le fichier à la fin.

```
/**
* Takes a pointer to a bmp struct and a filename
* It opens the file calls bmp_load_header which
* loads in the header, then verify that its a BMP
* file, it asserts if its not. otherwise it proceeds
* to call bmp_load_dib_header and then bmp_load_pixels.
* Then it finally closes the file
*/
void bmp_load(bmp* bmp, const char* filename);
```

2- Implémentez la fonction `bmp_load_header(bmp_header* header, FILE* file)` qui remplit la structure `bmp_header` avec les données correspondantes du fichier `file`.

```
/**
* Takes a pointer to the bmp_header and a FILE pointer
* It read relevant data from the file and fills the relevant
* bmp_header fields with the necessary data
*/
void bmp_load_header(bmp_header* header,  FILE* file);
```

3- Implémentez la fonction `bmp_load_dib_header(bmp_dib_header* dib_header, FILE* file)` qui remplit la structure `bmp_dib_header` avec les données correspondantes du fichier `file`.

```
/**
* Takes a pointer to the bmp_dib_header and a FILE pointer
* It read relevant data from the file and fills the relevant
* bmp_dib_header fields with the necessary data
*/
void bmp_load_dib_header(bmp_dib_header* dib_header, FILE* file);
```

4- Implémentez la fonction `bmp_load_pixels(const bmp_dib_header* const dib, unsigned** pixels, FILE* file)` qui charge toutes les données de pixels du fichier `file` dans la matrice `pixels`.

```
/**
* Takes a pointer to the bmp_dib_header, a pixels matrix and a FILE pointer
* It read the pixel data from the file and then it fills the pixels matrix
*/
void bmp_load_pixels(const bmp_dib_header* const dib, unsigned** pixels, FILE* file);
```

5- Libère la mémoire utilisée dans le struct `bmp`

```
/**
* Liberates the memory used in the bmp object
*/
void bmp_destroy(bmp* bmp);
```

6- Il est temps de consolider et de tester votre travail. Compilez ce que vous avez fait et exécutez les tests en utilisant respectivement les commandes `make` et `make run`. Si vous avez réussi la deuxième ensemble de tests (`Test Set 2`), félicitations, vous pouvez passer à la partie suivante.

Partie 3 : Traitement d'image, convolution

Cette dernière partie du TD nous mettra en œuvre une fonction liée à l'image facilitant son traitement.
1- Implémentez la fonction `init_image(unsigned width, unsigned height)` qui initalise l'objet image et alloue la matrice des pixels.

```
/**
* Initialize an image by setting width and height
* and allocating the colors matrix
*/
image init_image(unsigned width, unsigned height);
```

2- La fonction `convert_to_image(const bmp* const bmp)` convert la `struct bmp` en `struct image`. La séparation des canaux de couleur doit se produire dans cette fonction (Aide : utilisez la fonction `clz` et les masques de bits RGB qui existent dans l'en-tête dib)

```
/**
* Takes a bmp const pointer and converts it to an image object
* In order to do this the width, height and the pixels matrix
* must be copied over to the image struct
*/
image convert_to_image(const bmp* const bmp);
```

3- Cette fonction doit copier l'image passée en argument et renvoyer la copie

```
/**
* Copy an existing image to another one
*/
image copy_image(const image* const img);
```

4- Convertir une image normale en une image en niveaux de gris cela se fait en prenant la moyenne ds composants `r`, `g`, `r`.

```
/**
* Converts an image to grayscale this can be done by averaging
* the R, G and B component of each pixel
*/
void grayscale_image(image* img);
```

5- Effectue le produit de convolution entre le `kernel` (qui est une matrice de dimensions `kernel_size`) et l'image `src`, et stocke le resultat dans l'image `dst`.

```
/**
* Applies a convolution product between the source image and the kernel
* It stores the result in the destination image
*/
void apply_filter(image* dst, image* src, unsigned kernel_size, float* kernel);
```

6- Enregistre l'image passée en argument sur le disque. Il crée un fichier appelé nom et stocke les pixels de couleur dans un format BGRA (ce qui signifie que le composant `B` est écrit en premier, puis `G` puis `R` et enfin `A`).

```
/**
* Converts an image to grayscale this can be done by averaging
* the R, G and B component of each pixel
*/
void save_raw(const image* const img, const char* name);
```

7- Libère la mémoire utilisée par l'image

```
/**
* Liberates the memory related to the image object
*/
void destroy_image(image* img);
```

6- Compilez ce que vous avez fait et exécutez les tests en utilisant respectivement les commandes `make` et `make run`. Si vous avez réussi la troisième ensemble de tests (`Test Set 3`), félicitations, vous pouvez passer à la partie suivante.

Partie 4 : Celebrate

1- Il est temps de célébrer et de voir ce que vous avez fait exactement. Visitez ce site Web et téléchargez l'image que vous avez générée précédemment. Afin de voir les résultats, assurez-vous d'avoir les bonnes configurations.

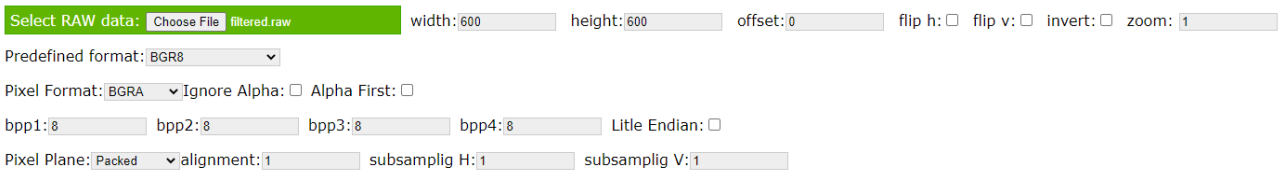

FIGURE 1 – Les configurations que vous devez utiliser

2- Essayez d'écrire un programme qui applique un autre filtre (il suffit de changer le noyau) sur la même image et observez le résultat. Pour trouver des noyaux intéressants, visitez cette page wikipedia.
3- Si vous avez encore de l'énergie, implémentez une fonction qui enregistre la structure `image` sous forme de fichier `bmp`. De cette façon, vous pouvez visualiser le résultat directement sur votre ordinateur.

Aide avec les fichiers

```
// FILE* fopen (const char* filename, const char* mode);
// Opens a file with the name filename, the mode specify the file access mode. It can be:
// "r": read, used for reading existing files
// "w": write, create an empty file to write to. If the file exists then its content is overwritten
// "a": append, opens a file then it writes data at the end of it. The file is created if it does not exist
// Appending b to the mode will open the file as a binary file.
// NOTE: in our case here we will be using binary files, processing text files isn't that different.

// fseek(FILE* file, long int offset, int origin);
// Sets the position indicator associated with the file to a new position.
// origin can be:
//      - SEEK_SET : Beginning of file
//      - SEEK_CUR : Current position of the file pointer
//      - SEEK_END : End of file
// Offset :
//      - Binary files: Number of bytes to offset from origin.
//      - Text files: Either zero, or a value returned by ftell.

// size_t fread (void * ptr, size_t size, size_t count, FILE* file);
// Pointer to a block of memory with at least size*count bytes
// size : size of each elemnt to be read
// count : number of elements to be read, each must be with size of size bytes

// size_t fwrite (const void* ptr, size_t size, size_t count, FILE* stream);
// ptr : pointer to the elments to be written
// size : size in bytes of each elemnt to be written
// count : number of elements each one with size of size bytes
```