université de BORDEAUX

Écriture de programmes utilisant la bibliothèque LGPL `libimago` de lecture et écriture d'images

DESCRIPTION DES ACTIVITÉS

Présentation du module

Description générale du travail attendu : mettre en œuvre différentes opérations en lien avec la représentation spatiale des images en utilisant la bibliothèque `libimago` écrite en langage C sous licence LGPL.

Activité notée : écriture de programmes en langage C utilisant la bibliothèque LGPL `libimago` :
- `img-format.c` : chargement et identification du mode de codage d'un fichier image ;
- `img-convert.c` : recopie d'un fichier image (le cas échéant sous un format différent) ;
- `img-rotchan.c` : rotation des canaux rouges, verts et bleus d'une image RGB ;
- `img-comp.c` et `img-decomp.c` : composition et décomposition des canaux d'une image RGB ;
- `img-gummira.c` : création de l'image d'une fonction (attracteur de Gumowski Mira).

Modalités : activité effectuée en binôme sur une durée de deux semaines.

Modalités d'évaluation de l'activité

Système de notation. Cette activité est évaluée par référence à cinq niveaux d'apprentissage :
- **échec** : aucun élément de réponse correct n'a été fourni ;
- **insuffisant** : la réponse contient des éléments attendus mais n'atteint pas le niveau minimum requis ;
- **suffisant** : un minimum requis d'éléments attendus est présent et correct ;
- **bon** : les éléments attendus sont globalement présents et corrects ;
- **excellent** : la réponse est complète et traduit une maîtrise des éléments évalués dans le contexte de cette activité.

Les notes correspondantes sont respectivement F, D, C, B et A. Les notes A, B et C des trois premiers niveaux peuvent le cas échéant être modulées en A+, A-, B+, B-, C+ et C-.

Critère d'évaluation. Capacité à utiliser la repésentation spatiale des images dans un programme en langage C (lecture et écriture, accès aux pixels, manipulation des canaux, fonction image)

Indicateurs d'appréciation du niveau atteint.

D : les programmes contiennent les bons appels de fonction et compilent correctement sous un environnement standard ISO C99/POSIX (par exemple en utilisant la commande `gcc`) mais ne produisent pas de résultats corrects sur les jeux d'essais.

C : le niveau D est atteint et les programmes `img-convert`, `img-format` et `img-rotchan` fonctionnent correctement sur les jeux d'essais ;

B : le niveau C est atteint et les programmes `img-comp` et `img-decomp` fonctionnent correctement sur les jeux d'essais ;

A : le niveau B est atteint et le programme `img-gummira` fonctionne correctement sur les jeux d'essais ;

Les jeux de test proposés pour chaque activité sont similaires à ceux qui seront utilisés pour évaluer les réponses.

1 Traduction en C de différents codages de pixels

— Les différents modes de codage supportés par la bibliothèque sont définis par l'énumération `enum img_fmt` dont les valeurs sont :

```
IMG_FMT_GREY8
IMG_FMT_RGB24
IMG_FMT_RGBA32
IMG_FMT_GREYF
IMG_FMT_RGBF
IMG_FMT_RGBAF
```

— Ils permettent de représenter des images comportant un seul canal (`GREY`), trois canaux (`RGB`) ou quatre canaux (`RGBA`).

— Chaque canal est codé soit par un entier sur huit bits (les trois premières constantes) soit par un nombre flottant (les trois dernières).

Activité **1** *En utilisant les constructions `char`, `float` et `[]` du langage C, écrire les types qui correspondent à chacun de ces modes de codage.*

2 [noté] Recopie d'un fichier image (`img-convert.c`)

— Le codage interne d'une image par la bibliothèque `libimago` est décrit par la structure :

```
struct img_pixmap
{
    void *pixels;
    int width;
    int height;
    enum img_fmt fmt;
    int pixelsz;
    char *name;
};
```

— Les quatre premier champs contiennent respectivement l'adresse du buffer contenant les pixels de l'image, la largeur et la hauteur de l'image et son mode de codage (décrit question suivante) ;

— Les deux fonctions suivantes permettent d'allouer dynamiquement une structure représentant une image vide et de la libérer :

```
struct img_pixmap *img_create(void)
void img_free(struct img_pixmap *img);
```

— Les deux fonctions suivantes permettent de charger en mémoire le contenu d'un fichier image de nom `fname` au moyen d'une structure `struct img_pixmap` préalablement crée, et réciproquement de sauver une image en mémoire dans un fichier :

```
int img_load(struct img_pixmap *img, const char *fname); /* error: -1 */
int img_save(struct img_pixmap *img, const char *fname); /* error: -1 */
```

Activité **2**

Au moyen des fonctions
- *img_create,*
- *img_load*
- *img_save*

écrire le fichier C `img-convert.c` implémentant la commande d'usage :

Usage: `img-convert INPUT_IMAGE OUTPUT_IMAGE`

et recopiant le fichier `INPUT_IMAGE` dans le fichier `OUTPUT_IMAGE`.

Jeux de test. La programme devra être testé avec différentes conversions de fichiers dont les noms d'entrée et de sortie peuvent être suffixés par `.png`, `.jpg`, `.jepg`, `.ppm` ou `.pgm`.

3 [noté] Identification du mode de codage d'une image (`img-format`)

— Les trois fonctions suivantes permettent d'obtenir des informations sur le mode de codage :

```
int img_is_float(struct img_pixmap *img);
int img_has_alpha(struct img_pixmap *img);
int img_is_greyscale(struct img_pixmap *img);
```

Activité **3**

Sachant que la bibliothèque `libimago` utilise la representation interne réel flottant pour charger une image codée sur 16 bits par canal, écrire à l'aide des fonctions
- *img_create,*
- *img_load,*
- *img_is_greyscale,*
- *img_is_float,*
- *img_has_alpha,*
- *img_free*

le fichier C `img-format.c` implémentant la commande d'usage :

Usage: `img-format INPUT_IMAGE...`

*affichant pour chaque fichier **F** image reçu en argument une des informations suivantes :*

*"**F**": grayscale 8 bits image*
*"**F**": grayscale 16 bits image*
*"**F**": RGB 8 bits image without alpha channel*
*"**F**": RGB 8 bits image with alpha channel*
*"**F**": RGB 16 bits image without alpha channel*
*"**F**": RGB 16 bits image with alpha channel*

ou un message d'erreur :

`img-format: cannot load "F"`

Jeux de test. La programme peut être testé avec les huit fichiers fournis (images de damier).

4 [noté] Rotation des canaux d'une image RGB (`img-rotchan.c`)

— Les deux fonctions suivantes permettent de charger dans un buffer l'image contenue dans un fichier image et de recopier le contenu d'un buffer codant une image fichier image.

— Le correspondance entre le format externe (sur fichier) et le format interne (dans le buffer) est spécifié par un paramètre de type `enum img_fmt`.

— Ces fonctions n'utilisent donc pas le type `struct img_pixmap`.

```
void *img_load_pixels(const char *fname, int *x_size, int *y_size,
                      enum *img_fmt format);

int img_save_pixels(const char *fname, void *pix, int x_size, int y_size,
                    enum img_fmt format);
```

— La fonction `img_load_pixels` lit l'image contenue dans le fichier `fname` et retourne un pointeur vers un vecteur de pixels du type spécifié par `enum img_fmt` (voir aussi question 4) et la dimension de l'image est retournée au moyen des pointeurs `x_sixe` et `y_size`.

— En cas d'erreur la fonction `img_load_pixels` retourne `NULL`.

— La fonction `img_save_pixels` écrit dans le fichier `fname` l'image d'adresse mémoire `pix`, de taille `x_sixe` × `y_size` et de format interne `format`.

Activité 4

Au moyen des fonctions

— *`img_load_pixels`,*

— *`img_save_pixels`*

et sans utiliser d'autre fonction de la bibliothèque `libimago`, écrire le fichier C `img-rotchan.c` implémentant la commande d'usage :

Usage : img-rotchan INPUT_IMAGE OUTPUT_IMAGE

Cette commande charge le contenu de l'image `INPUT_IMAGE` en utilisant le format interne `IMG_FMT_RGB24`, effectue une rotation des canaux RGB et stoque le résultat dans le fichier `OUTPUT_IMAGE`. Cette rotation consiste à envoyer le canal rouge de l'image source sur le canal vert de l'image résultat, et respectivement le vert sur le bleu et le bleu sur le rouge.

Jeux de test. Le programme peut être testé avec les trois fichiers fournis (images de vitraux). La rotation des canaux du premier fichier doit produire le second, du second le troisième, et du troisième le premier.

5 [noté] Décomposition et recomposition des canaux d'une image RGB

On définit la fonction C

```
char *channel_extract(char *pixels, int size, int channel_number)
```

qui reçoit un pointeur sur une image RGB de taille `size` et codée sur huit bits par canal (paramètre `pixel`) et renvoie un pointeur sur une image monochrome allouée dynamiquement, de même taille, et correspondant au canal de numéro `channel_number` (0 pour rouge, 1 pour vert et 2 pour bleu).

On définit symétriquement la fonction C

```
char *channel_compose(char *r_chan, char *g_chan, char *b_chan, size_t size)
```

qui alloue dynamiquement une image RGB de taille `size`, codée sur huit bits par canal, l'initialise avec les trois canaux codés sur 8 bit et de même taille `r_chan` (canal rouge), `g_chan` (canal vert) et `b_chan` (canal bleu), et retourne un pointeur sur cette image.

Activité **6**

Écrire les fonctions `channel_extract` et `channel_compose` et les utiliser pour implémenter les commandes :

```
img-decomp INPUT_IMAGE RCHAN_OUTPUT GCHAN_OUTPUT BCHAN_OUTPUT
img-comp RCHAN_INPUT GCHAN_INPUT BCHAN_INPUT OUTPUT_IMAGE
```

La commande `img-decomp` produit trois fichiers monochromes contenant les canaux repectivement rouge, vert et bleu de l'image RGB `INPUT_IMAGE`.

Réciproquement, la commande `img-comp` reçoit trois fichiers monochromes (`RCHAN_INPUT`, `GCHAN_INPUT` et `BCHAN_INPUT`) et produit une image RGB formée à partir de ces trois canaux.

Vérifier qu'en enchainant les deux commandes à partir d'une image RGB, on retrouve bien cette image.

Jeux de test. Le programme `img-comp` peut-être testé avec les exemples d'acquisition multibande présentés en cours (diapositives 17 et 18) et dont les fichiers sont fournis. Le programme `img-decomp` pourra être testé combiné à `img-comp` pour reproduire le résultat de rotations des canaux de l'activité précédente.

6 [noté] Création de l'image d'une fonction mathématique (`img-gummira.c`)

— La fonction `img_set_pixels` permet d'allouer un buffer à une structure `struct img_pixmap` précédemment initialisée (si elle possédait déjà un buffer image, il est libéré).

```
int img_set_pixels(struct img_pixmap *img,
                   int x_size, int y_size, enum img_fmt format, void *pix);
```

— La taille du buffer est déduite des paramètres `x_size`, `y_size` et `format`.

— Si le pointeur `pix` est non nul, il doit pointer vers un buffer d'image de la taille et du format requis et son contenu est recopié dans le buffer alloué.

— En cas d'erreur la fonction renvoie -1.

— Les deux fonctions suivantes permettent de récupérer ou de modifier le contenu d'un pixel de l'image.

```
void img_setpixel(struct img_pixmap *img, int x, int y, void *pixel);
void img_getpixel(struct img_pixmap *img, int x, int y, void *pixel);
```

Activité **7**

On définit l'attracteur de Gumowski-Mira au moyen de la suite de points (x, y) du plan :

$$\begin{aligned} x_{i+1} &= b * y_i + f(x_i, a); \\ y_{i+1} &= -x_i + f(x_{i+1}, a); \end{aligned}$$

À l'aide des fonctions

- `img_create()`,
- `img_set_pixels()`,
- `img_setpixel()`.
- `img_save()`

écrire le fichier C `img-gummira.c` *implémentant la commande d'usage :*

`Usage: img-gummira WIDTH A B X_0 Y_0 N OUTPUT_IMAGE`

créant une image carrée de largeur et hauteur `WIDTH`*, au format RGBA, et visualisant les* `N` *premiers points de l'attracteur de Gumowski-Mira calculé pour la fonction $f(x, a) = ax + \frac{2(1-a)x^2}{1+x^2}$ et à partir des valeurs initiales* `X_0` *et* `Y_0`*.*

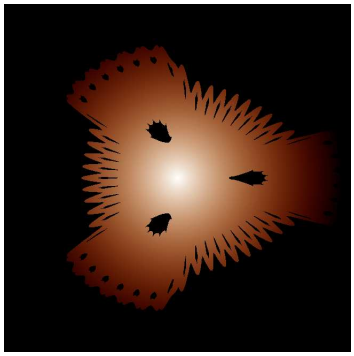
On donnera à chaque point (x_i, y_i) de la suite $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ la couleur de coordonnées RGBA $(255, \alpha \times 255, \alpha^2 \times 255, \alpha^2 \times 255)$ avec

$$\begin{cases} \alpha &= 1 - \frac{1}{\sqrt{2}} \sqrt{\left(\frac{x_i}{M}\right)^2 + \left(\frac{y_i}{M}\right)^2} \\ M &= \max\left\{ \max\{|x_1|, \ldots, |x_N|\}, \max\{|y_1|, \ldots, |y_N|\} \right\}. \end{cases}$$

On appliquera un facteur d'échelle calculé en fonction de WIDTH, en faisant correspondre le domaine de l'image aux valeurs de la fonction comprises entre $-M$ et $+M$, de manière à remplir au mieux l'image.

Jeux de test. Pour tester le programme `img-gummira`, on pourra utiliser les valeurs du tableau suivant en paramètre (voir un exemple d'image visualisée sur fond noir, obtenue pour 20 000 000 d'itérations avec les valeurs de la première ligne du tableau).

a	b	x_0	y_0
$-.5355$	1.0	1.0	1.0
$-.4849900881$	0.9758744406	-7.51072037	$13,66255124$
-0.75039721	1.0	-5.0	-11.0
.266865349	1	$-.7921202843$	-0.16530339760
$-.869471195$	1	1	15
$-.7842208082$	0.9890020033	4.08073727	12.21132774
-0.6285	1	-5.2	-9
.368164112	1	5.64147642	2.07640240
-0.6286990697	1	4	6
-0.4241828808	0.951219909	6.03780286	-5.70218011



Ces données sont fournies avec un script shell pour simplifier leur utilisation. Différents exemples sont disponibles dans un fichier PDF.

Annexe : utilisation de la bibliothèque libimago

Installation.

— La bibliothèque `libimago` est développée par John Tsiombikas sous licence LGPL. Elle est toujours maintenue et disponible sur github. Elle est en pricipe multiplateforme mais choisissez de l'installer sous GNU/Linux si vous souhaitez pouvoir bénéficier d'une assistance dans le cadre de cet enseignement. Il est également nécessaire d'installer les bibliothèques `libpng` et `libjepg` dont les paquets sont disponibles sous les principales distributions GNU/Linux.

— Étapes de l'installation de `libimago` :

```
git clone https://github.com/jtsiomb/libimago
cd libimago
./configure
./make
sudo make install
```

— Par défaut sous GNU/Linux, les fichiers nécessaires à la mise en œuvre de la bibibliothèque sont installés à partir du répertoire `/usr/local` (sous-répertoires `include` et `lib`). Ils peuvent être désinstallés en exécutant `make uninstall`. Vous pouvez également installer ces fichiers dans un répertoire de votre choix en précisant son chemin en paramètre de la commande `configure`. Exemple :

```
./configure --prefix=$HOME/local
./make
./make install
```

Il est dans ce cas nécessaire d'indiquer ce chemin au compilateur au moyen des options `-I` et `-L`. Exemple :

```
-I$HOME/local/include
-L$HOME/local/lib
```

Pour faire exécuter un programme ainsi compilé, il peut être nécessaire de définir dans l'environnent Unix le chemin des bibliothèques partagées. Avec les mêmes conventions que précédemment, cela se fait en initialisant la variable d'environnement `LD_LIBRARY_PATH` depuis le shell :

```
export LD_LIBRARY_PATH=$HOME/local/lib
```

Vérification de l'installation.

— La compilation d'un programme utilisant la bibliothèque se fait avec les options d'édition de lien

`-limago -lpng -ljpeg`

— L'interface avec la biliothèque est contenue dans le fichier en-tête `imago2.h`.

— Pour tester la bonne installation de la bibliothèque il suffit de compiler (sans erreur) un programme contenant les lignes :

```
#include <imago2>
[ ... ]
struct img_pixmap *img = img_create(void);
[ ... ]
```