Floating images processing

tTh

3 septembre 2020

1 Image flottante ?

Mais de quoi parle-t-on exactement ?

Traditionnellement, les valeurs des pixels dans les images informatiques sont mémorisées sur 8 bits, un octet, soit 256 valeurs différentes. Ceci dit, on trouve parfois des images codées sur 16 bits par composante, mais c'est loin d'être le cas général. J'ai donc souhaité aller plus loin, et coder chaque canal de chaque pixel en virgule flottante sur 32bits, le type `float` du langage C. Ce qui correspond à la norme IEEE 754-1985.

Attention, tout le code que nous allons voir ensemble est en perpétuelle évolution [1], et sa fiablité (surtout sur certains aspects mathématiques) reste à démontrer. Mais le service après-vente est assez réactif. Du moins pour ceux qui suivent le canal `#tetalab` sur le réseau IRC de Freenode.

Attention ! ce document commence par une bonne rafale de technique parfois hardue [2]. Vous avez parfaitement le droit de sauter directement à la page 11 pour quelque chose de plus concret.

Table des matières

1. voir page 14
2. gni ?

2 Théorie

Pour le moment, seule la quête de l'empirisme absolu a été visée. Les justifications mathématiques attendront le retour du schmod777. Ceci dit, rien ne nous empêche d'aller consulter Wikipedia :

> An IEEE 754 32-bit base-2 floating-point variable has a maximum value of $(2 - 2^{-23}) \times 2^{127} \approx 3.4028235 \times 10^{38}$. All integers with 7 or fewer decimal digits, and any 2^n for a whole number $-149 \leq n \leq 127$, can be converted exactly into an IEEE 754 single-precision floating-point value.
>
> In the IEEE 754-2008 standard, the 32-bit base-2 format is officially referred to as binary32; it was called single in IEEE 754-1985.

Ce qui nous conduit à estimer qu'il est possible de cumuler environ quelques milliers d'images standard à 256 niveaux, sans trop avoir à se soucier des éventuelles pertes de précision. Mais ça demande à être confirmé par des esprits supérieurs.

2.1 Dynamique

Dynamique, précision et *macheps* ?

2.2 Pixel négatif ?

Il est très difficle d'imaginer une lumière négative. Sauf peut-être si nous songeons à des coefficients d'absorption ?

3 Premier exemple

FloatImg a débuté sous la forme de quelques fonctions basiques en C, gérant la structure des données d'image en mémoire et sur disque. Ça a été imaginé de façon presque empirique, mais nous sommes tous là pour améliorer les choses, dans la mesure de nos moyens. Nous allons donc directement rentrer au cœur du problème, en écrivant quelques lignes de code.

Pour commencer par quelque chose de simple, nous allons créer une image RGB complètement noire, puis l'enregistrer dans un fichier `.fimg`, un format complètement inconnu, puisque je viens de l'inventer à l'instant même.

Tout d'abord, nous devons déclarer et garnir quelques variables pour gérer la machinerie interne.

```
int             width = 640, height = 480;
char            *fname = "exemple.fimg";
FloatImg        fimg;
```

Ensuite, nous enchainerons trois étapes : création de l'image en mémoire centrale, initialisation des valeurs de chaque pixel à 0.0, et pour conclure, enregistrement dans un fichier [3] binaire.

```
foo = fimg_create(&fimg, width, height, FIMG_TYPE_RGB);
if (foo) {
        fprintf(stderr, "create floatimg -> %d\n", foo);
        exit(1);
        }
fimg_clear(&fimg);
foo = fimg_dump_to_file(&fimg, fname, 0);
```

3. Au format ésotérique, mais très véloce.

```
if (foo) {
        fprintf(stderr, "dump fimg -> %d\n", foo);
        exit(1);
        }
```

Une fois ce code enrobé dans un `main()`, compilé et exécuté, nous pouvons entrevoir, grâce au logiciel `fimgstats` (voir page 12), le résultat sous forme de chiffres divers, et/ou inutiles :

```
$ ./fimgstats quux.img
----------- numbers from 'quux.img' :
  640    480   3  0x7f3718c4f010 0x7f3718d7b010 0x7f3718ea7010
surface      307200
mean values:
      R         0.000000
      G         0.000000
      B         0.000000
      A         0.000000
max value     0.000000
```

Nous avons donc sous la main une mécanique qui ne demande qu'à faire des trucs futiles et des images qui clignotent. La suite vers la page 5.

Vous trouverez dans le répertoire `tools/` d'autres exemples de mise en œuvre des fonctions disponibles sous formes d'outils en ligne de commande, lesquels sont décrits en page 11.

4 Installation

Sauf indications contraires, ces instructions se réfèrent à une distribution Debian récente.

Attention, ça devient un peu gore. Myrys, punk, toussa...

4.1 Prérequis

Vous devez, en dehors des outils classiques (bash, gcc, make...), avoir quelques bibliothèques installées[4] : libv4l2, libpnglite, libtiff, libnetpbm[5], libz[6], libcurses, libcfitsio-dev éventuellement avec le -dev correspondant, et probablement d'autres choses.

Il est même quasiment certain que Bash soit indispensable, tout comme GNU/make. Une connaissance de base de l'utilisation du shell et de l'écriture de Makefile's sera un plus.

Il faut aussi savoir où trouver le code.

4.2 Compilation

Un script `build.sh` permet de construire approximativement le bouzin. Il est loin d'être parfait[7]. Dans chacun des répertoires à traiter, ce script devrait trouver un Makefile et un fichier `t.c` qui est le source de la cible par défaut du make.

Pour le moment, la procédure d'installation est un peu rude, pour ne pas dire clairement sommaire. Si le résultat de l'étape compilation vous semble correct, vous pouvez copier les deux fichiers `floatimg.h` et `libfloatimg.a` dans un emplacement approprié, par exemple `/usr/local/include` et `/usr/local/lib`.

4. Les `-dev` en plus pour Debian et dérivées
5. package libnetpbm10-dev
6. package zlib1g-dev
7. Il doit être possible de faire un Makefile récursif, mais...

Le script `install.sh`, à la racine du projet, est censé faciliter un peu la chose. Il prend également en compte la copie des divers binaires du dossier `tools/` (cf page 11) dans le répertoire prévu à cet effet : `/usr/local/bin`.

Il reste enfin quelques exemples d'utilisation des outils de la ligne de commande depuis un shell dans le répertoire `scripts/`. Ils sont décrits plus en détail page 14, et c'est à vous de les adapter à votre *usecase*. Faites-en ce que vous voulez.

5 Utilisation coté codeur

Classiquement, il y a un fichier `.h` à inclure dans chacun de vos codes source, `floatimg.h`, généralement logé dans `/usr/local/include` contenant un certain nombre de définition de structures, de macros, de constantes [8] et les prototypes des fonctions utilisables par vos logiciels.

Au niveau du code source, ces fonctions sont approximativement classées en deux catégories : `lib/` et `funcs/`. La première contient les choses qui sont relativement figées, et la seconde celles qui risquent de bouger. Cette classification est en fait arbitraire.

5.1 Structures, macros...

Les pixels flottants d'une image résidant en mémoire centrale sont décrits par un ensemble de données (certains appelent ça des *metadatas*) regroupées dans une jolie structure que nous allons examiner dès maintenant.

```
/*      in memory descriptor */
typedef struct {
        int             width;
        int             height;
        int             type;
        float           fval;
        int             count;
        float           *R, *G, *B, *A;
        int             reserved;
        } FloatImg;
```

Les deux premiers champs sont *obvious*. Le troisième est le type d'image : pour le moment, il y en a un certain nombre qui sont définis [9] : gris, rgb et rgba. Les constantes adéquates sont dans `floatimg.h`

```
#define FIMG_TYPE_GRAY              1
#define FIMG_TYPE_RGB               3
#define FIMG_TYPE_RGBA              4
#define FIMG_TYPE_RGBZ              99
```

Un peu plus loin, nous avons les pointeurs vers les différents *pixmaps* de l'image. En principe l'organisation interne de ces zones est improbable, puisque elle dérive d'idées approximatives. C'est cette utilisation constructive de larache qui fait que seuls les champs documentés de cette structure ne sont pas explosifs.

Mais revenons aux choses sérieuses... Les deux champs suivants (fval et count) sont à la disposition du *yuser* qui peut jouer avec à loisir pour faire, par exemple, ce genre de chose : imaginons un périphérique de capture qui nous fournisse des images en gris sur 4 bits (et linéaire). Et que nous voulions cumuler quelques images...

Le champ *count* sera mis à 0 et le champ *fval* sera initialisé à 15.0 (qui est la valeur maximale que peut renvoyer le capteur). Ensuite, dans la boucle capture/cumul, *count* sera incrémenté à

8. À l'ancienne, via le pré-processeur
9. et plus ou moins bien gérés...

chaque passe, et nous aurons donc, en finale, toutes les informations nécessaires pour exploiter au mieux la dynamique de notre image dans les étapes ultérieures, puisque la valeur maximale théorique est égale à $fval * count$.

La fonction `fimg_printhead(FloatImg *h)` affiche sommairement le contenu de ce descripteur, et `fimg_describe(FloatImg *head, char *txt)` propose un affichage plus détaillé. Ça aide parfois.

Une bonne partie des fonctions que nous allons voir est indéterministe. Ce qui veut dire, en langage de tous les soirs, que ça risque de ne pas être la même chose dans l'avenir.

5.2 Les fondations

La première chose que nous devons absolument voir est la gestion dynamique de la mémoire qui sera occupée par tous ces pixels flottants, ce qui est un sujet parfois délicat [10]. Elle est donc faite, à la base, par ces deux fonctions :

```
int fimg_create(FloatImg *fimg, int w, int h, int type);
int fimg_destroy(FloatImg *fimg);
```

L'appelant doit lui-même gérer le descripteur d'image (une structure C décrite plus haut) en le considérant comme un type semi-opaque dont la forme peut varier. Certains membres de cette structure sont documentés dans ce document, et les autres sont dangereux à toucher. Les types d'images actuellement gérés sont les trois grands classiques : gray, rgb et rgba. et expliquées quelques lignes plus haut.

Comme vous allez le voir plus loin, il y a plein de fonctions qui prennent en argument deux images : une source et une destination. Et dans la plupart des cas, ces deux images doivent être compatibles, c'est à dire même type et mêmes dimensions.

```
/* return 0 if compatible */
int fimg_images_not_compatible(FloatImg *a, FloatImg *b);
```

C'est bien beau d'être enfin résident en mémoire centrale, mais pouvoir aussi exister à long terme en étant stocké dans la matrice est tout aussi pertinent. Il y a deux opérations qui supportent le reste des transits ram/ps. Le format des fichiers est décrit page 9.

```
int fimg_dump_to_file(FloatImg *fimg, char *fname, int notused);
int fimg_load_from_dump(char *fname, FloatImg *where);
```

Recharger une image depuis un fichier nécessite que celle-ci et l'image de destination en mémoire ait précisément les mêmes caractéristiques (taille, type...), donc l'image en ram doit être pré-allouée. On peut connaitre ces valeurs en appelant `int fimg_fileinfos(char *fname, int datas[3])`.

Si tout s'est bien passé (valeur retournée égale à 0), on va trouver la largeur dans `datas[0]`, la hauteur dans `datas[1]` et le type dans `datas[2]` [11].

Je sais aussi que certains d'entre vous aiment la facilité, aussi je vais vous révéler l'existence d'un nouveau truc bien plus simple, une fonction qui enchaine ces deux actions (allocation, puis lecture), et s'utilise comme ça :

```
FloatImg        head;
memset(&head, 0, sizeof(FloatImg));
foo = fimg_create_from_dump("lena.fimg", &head);
```

Si la valeur retournée est différente de 0, c'est que quelque chose s'est mal passé. Certains messages sont parfois explicites.

10. GC or not GC ?
11. La fonction `fimg_type_is_valid(int type)` peut vous aider...

5.3 Dessiner

Bon, vous avez une image latente, et vous souhaitez dessiner dessus (ou dedans ?) avec vos encres flottantes ? Il y a des fonctions pour ça, par exemple :

```
int fimg_plot_rgb(FloatImg *head, int x, int y, float r, float g, float b);
```

Les paramètres sont explicites, mais leur validité doit être sévèrement controlée par l'appelant. Il y a une fonction soeur, `fimg_add_rgb`, qui ajoute du rgb à un pixel, laquelle fonction a d'ailleurs été à la base de la seconde génération de la photographie en cumul.

Inversement, la fonction `fimg_get_rgb(FloatImg *head, int x, int y, float *rgb)` permet de lire les valeurs des trois canaux d'un pixel donné. Là aussi, il n'y a aucun contrôle sur la validité des valeurs x et y de la demande.

Quand au canal *alpha*, il est pour le moment superbement ignoré. Ceci dit, on vient de me faire remarquer qu'il peut être utilisable aussi pour faire du *z-buffer* ...

5.4 Contraste

Certaines opérations d'ajustement du contraste d'une image semblent cohérentes avec la notion d'image flottante. Certaines d'entre elles, les plus simples, sont disponibles. Les autres sont à imaginer.

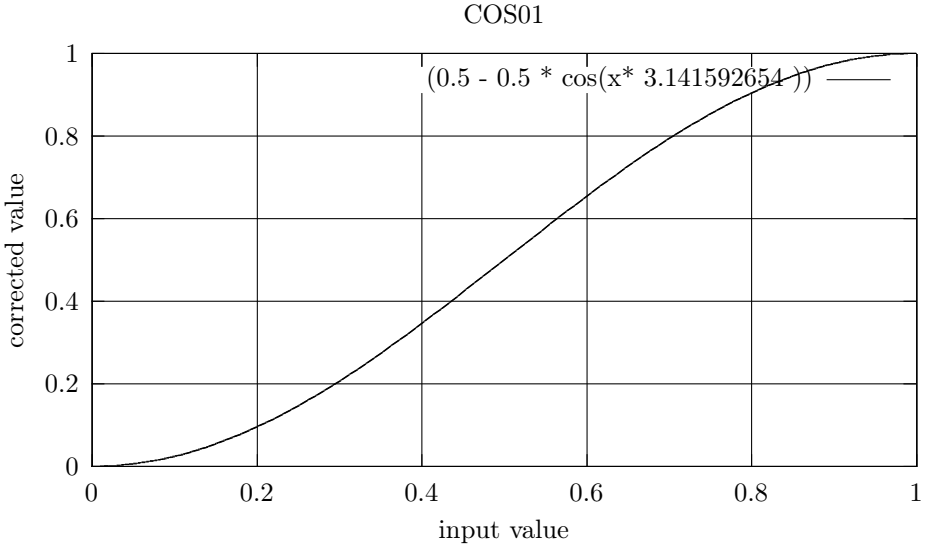

FIGURE 1 – Correcteur cos01

Ils prennent chacun trois paramètres, d'abord les images source et destination (`* FloatImg`), ensuite le troisième qui est un nombre en double précision donnant la valeur maximale *supposée* de l'image source, valeur qui peut être déterminée de plusieurs manières.

```
/* source in lib/contrast.c */
int fimg_square_root(FloatImg *s, FloatImg *d, double maxval);
int fimg_power_2(FloatImg *s, FloatImg *d, double maxval);
int fimg_cos_01(FloatImg *s, FloatImg *d, double maxval);
int fimg_cos_010(FloatImg *s, FloatImg *d, double maxval);
```

Si vous souhaitez rajouter votre propre méthode de modification de contraste, il y a quelques explication en page 11. Mais rien ne vous oblige à le faire.

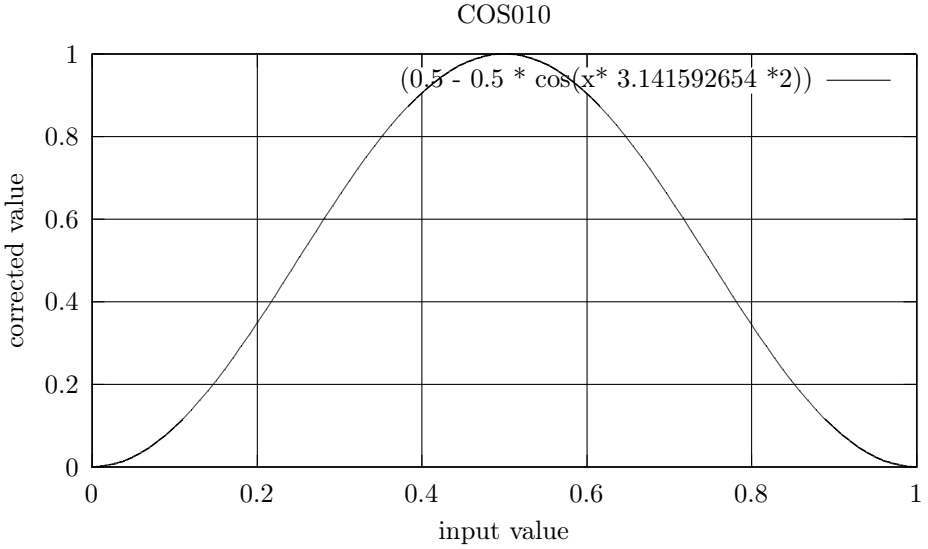

FIGURE 2 – Correcteur cos010

Rappelons qu'il est possible pour un logiciel applicatif comme `grabvidseq` (cf page 18) de renseigner deux champs du descripteur d'image avec des données pertinentes. Ces deux champs sont *fval* et *count*. Dans ce cas particulier, le premier indique la valeur maximale du capteur, et le second sert à compter le nombre de capture [12] effectuées.

La fonction `fimg_normalize(FloatImg *fi, double maxima, int notused);` tente de gérer ce cas d'utilisation. Son ajout au captureur d'images floues sera probablement le bienvenue. Je me suis bien rendu compte à l'usage [13] en situation festive qu'il manquait des données dans la chaine de traitement.

L'autre façon de procéder est d'explorer notre image à la recherche de la valeur maximale. La fonction `float fimg_get_maxvalue(&fimg)` est prévue pour ça de façon sommaire. C'est actuellement la méthode utilisée par l'outil qui sert à faire les modifications de contraste (page 13). On pourra aussi envisager d'utiliser `fimg_get_minmax_rgb(FloatImg *head, float mmvals[6])`, qui permet un contrôle bien plus fin des dérives.

La prochaine étape consistera à trouver une façon de faire une égalisation par histogramme qui respecte, dans toute sa futilité, le concept de pixel flottant. *Et c'est pas gagné...*

5.5 Géométrie

Très prochainement, le retour du blitter. Et pour attendre, un truc improbable, voire même inutile, en fait l'inverse de l'upscaling.

```
int fimg_halfsize_0(FloatImg *src, FloatImg *dst, int notused);
```

Attention lors de l'appel, le descripteur `dst` ne doit pas contenir d'image, et doit être effacé avec un bon `memset(&result, 0, sizeof(FloatImg));` bien senti. Et le résultat est très moyen : il n'y a pas d'interpolation.

```
int fimg_extract_0(FloatImg *src, FloatImg *dst, int x, int y);
```

Contrairement à la fonction précédente, celle-ci demande absolument une image de destination initialisée aux dimensions (largeur et hauteur) désirées.

12. Et c'est bien géré aussi dans l'upscaling.
13. Une histoire de *fonderie*, un logiciel censé faire des films flous à partir d'images floues

```
int fimg_rotate_90(FloatImg *src, FloatImg *dst, int notused);
```

Rotation de 90 degrés dans le sens horlogique [14] d'une image RGB. L'image de destination peut être soit vierge, soit pré-allouée aux bonnes dimensions (échange W et H).

5.6 Format du fichier FIMG

D'un design très empirique, c'est certainement à revoir pour l'avenir. Tout d'abord pour normaliser l'endianess et le packing...

```
typedef struct {
        char            magic[8];
        int             w, h, t;
        } FimgFileHead;
```

... Mais aussi pour faciliter l'ajout de métadonnées, telles que la valeur maximale, la date de création, une longueur d'onde, et bien plus encore.

5.7 Exportation & Importation

Notre format de fichier étant totalement inconnu, il nous faut bien exporter nos images en quelque chose de plus connu. Bien entendu, c'est toujours affaire de compromis entre précision de valeurs et taille des fichiers.

Et dans le sens inverse, il serait bien de savoir importer le monde extérieur dans nos sombres caves à pixel. Il faut quand même reconnaitre que c'est un peu la jungle dans les formats de fichiers d'image, ce qui explique le retard dans ce domaine...

5.7.1 Vers PNM

Nous avons ici 16 bits par composante, soit 65536 valeurs différentes, ce qui est bien au-delà de ce que peuvent percevoir vos yeux. Hélas, c'est au prix d'une taille énorme sur les fichiers. D'un autre coté, l'utilisation du codage ASCII (alors qu'on pourrait mettre du binaire, plus compact) y est pour quelque chose.

```
int fimg_save_as_pnm(FloatImg *head, char *fname, int flags);
```

Le bit `0` du paramètre `flags` mis à `1` demande à la fonction de faire la mise à l'échelle avec le couple *fvalue/count* décrit plus haut dans cette doc. Et si il est à zéro, c'est la fonction de recherche de valeur maximale (cf page 7) qui est utilisée.

Le bit `1` permettra bientôt de demander l'enregistrement de métadonnées pertinentes, telle que l'epochtime de l'enregistrement.

Les autres bits ne sont pas utilisés et doivent être à zéro.

5.7.2 Vers PNG

Actuellement, on peut enregistrer uniquement en mode RGB, 8 bits par composante, mais on a quand même une bonne compression, ça compense. J'utilise *libpnglite* avec qui j'ai un peu de mal à suivre. Mais je me soigne. Le mode 16 bits va bientôt arriver. On peut aussi songer à l'export de metadatas.

```
int fimg_save_as_png(FloatImg *src, char *outname, int flags);
```

Tous les flags doivent être à zéro. Sinon, ça foire parfois.

14. ou trigonométrique,le code et la doc ne semblent pas d'accord.

5.7.3 Vers TIFF

Le format canonique de la PAO du siècle dernier. Il permet de gérer une foultitude de formats numériques. C'est aussi un format classique proposé par les gros scanners corporates.

```
int fimg_write_as_png(FloatImg *src, char *outname, int flags);
```

Tous les flags doivent être à zéro.

5.7.4 Vers FITS

Ce format est essentiellement utilisé pour stocker des images d'astronomie, donc il peut aussi servir pour des images floues. Cette partie est basée sur la bibliothèque `cfitsio` de la NASA.

```
int fimg_save_R_as_fits(FloatImg *src, char *outname, int flags);
int fimg_save_G_as_fits(FloatImg *src, char *outname, int flags);
int fimg_save_B_as_fits(FloatImg *src, char *outname, int flags);
```

Bizarrement, l'image est stockée *upside-down* et je ne sais pas encore comment régler ce petit détail.

Tous les flags doivent être à zéro.

5.8 Utilitaires

Commençons par quelques petits trucs pour nous faciliter la vie dans des domaines annexes, tels que l'interprétation d'arguments dans la ligne de commande ou un fichier de configuration.

```
int parse_WxH(char *str, int *pw, int *ph)
int parse_double(char *str, double *dptr)
```

La fonction `int format_from_extension(char *fname)` examine un nom de fichier tel que `lena.xxx`, et retourne, si la partie `xxx` est connue, un éventuel nombre positif, dont les valeurs sont déclarées dans floatimg.h le valeureux. Les extensions actuellement connues sont : fimg, png, pnm, fits et tiff.

To be continued...

5.9 Effets

Quelques routines qui servent futilement à *brotcher* les images en tripotant les flux visuels chromatiques.

```
int fimg_killcolors_a(FloatImg *fimg, float fval);
int fimg_killcolors_b(FloatImg *fimg, float fval);
int fimg_colors_mixer_a(FloatImg *fimg, float fval);
```

5.10 Filtrages

Pour commencer, il faut que je réfléchisse au traitement des bordures des images. Ensuite que je débuggue ces deux fonctions :

```
int fimg_lissage_2x2(FloatImg *img);
int fimg_killborders(FloatImg *img);
```

Bon, oké, ça marche ? Passons à l'tape suivante. La convolution avec une matrice 3x3, c'est possible. Et pas trop compliqué à faire. Bon, il reste le souci avec les bordures, souci qui ne peut être que temporaire, mais ésotérique à fixer.

Passons maintenant aux choses sérieuses, et définissons la description d'un filtre 3x3.

```
typedef struct {
        float                   matrix[9];
        float                   mult;
        float                   offset;
        } FimgFilter3x3;
```

L'usage des champs `mult` et `offset` n'est pas clairement défini. Le prototype de la fonction de filtrage non plus, mais assez simpe quand même. Source et destination ne peuvent désigner la même image, et le champ `matrix` du filtre doit contenir des valeurs cohérentes.

```
int fimg_filter_3x3(FloatImg *src, FloatImg *dst, FimgFilter3x3 *filtr)
```

Comme dans la plupart des cas, la gestion des valeurs négatives de pixel est laissé au hasard. Quoique, il doit bien exister quelques solutions de contournement : clamping ou shift ?

To be continued...

5.11 Exemple de fonction

Nous allons maintenant écrire une fonction intégrable dans le répertoire `funcs/`. Elle aura donc accès aux *internals*[15] de FloatImg, une chose qui est en principe interdit aux programmes *enduser*. Soyez prudents.

Cette fonction va faire quelque chose à partir d'une image source et d'une valeur, et écrire le résultat dans une image de destination. Pour simplifier les choses, nous n'allons traiter que les images de type FIMG_TYPE_RGB, de loin le plus répandu par les temps qui courent.

```
int fimg_example(FloatImg *s, FloatImg *d, float value)
{
int     size, index;

if ( s->type!=FIMG_TYPE_RGB || d->type!=FIMG_TYPE_RGB) {
        perror("fimg_example");
        return -1;
        }

size = s->width * s->height;
for (idx=0; idx<size; idx++) {
        d->R[idx] = s->R[idx] - value;
        d->G[idx] = s->G[idx] - value;
        d->B[idx] = s->B[idx] - value;
        }

return 0;
}
```

Je vous laisse imaginer les dégats que peut faire cette fonction en utilisation réelle. Mieux, je vous propose d'essayer par vous-même. En particulier tout le reste du code qui suppose qu'un pixel ne peut **pas** être négatif va peut-être exploser de rire. Vous pouvez aussi remarquer qu'il n'y a pas de controle de cohérence sur les dimensions des deux images, malgré l'existence de fonctions prévues à cet effet..

6 Les outils

3615mavie : sur des projets comme celui-ci, qui travaillent in-fine sur des objets que l'on peut considérer comme « physiques », il est important de passer à une utilisation normale[16] et

15. que je peux décider de changer n'importe quand
16. Il y a une vie en dehors de git.

construire des trucs qui mettent en action le code primitif.

Ces machins ont en commun quelques options bien pratiques : `-h` pour avoir un résumé des options disponibles, `-v` qui augmente la puissance de bavardage, et `-K nn.nn` pour un paramètre flottant. Dans un avenir incertain, il existera des pages de man.

6.1 mkfimg

Propose la création d'un fichier contenant une image de « teinte » constante (ou pas). Cette notion de teinte est assez inconsistante pour le moment, mais ça n'est pas si grave que ça.

```
tth@debian:~/Devel/FloatImg/tools$ ./mkfimg -h
Usage:  mkfimg [options]  quux.fimg width height
        -k N.N  give a float parameter
        -t type         howto make the pic
                        black, drand48...
        -v      increase verbosity
```

La plupart des types d'image générée prennent un paramètre flottant qui devra être donné avec l'option `-k F.F` avec une valeur par défaut à 1.0.

black/gray/grey : efface avec 0.0 (black) ou avec la valeur `-k` (gray).

drand48 : beaucoup de bruit dans chacun des canaux.

hdeg/vdeg : dégradé du noir au blanc (relatif à `-k`).

6.2 png2fimg

Grosse panne à réparer.

```
tth@debian:~/TMP/floatimg$ png2fimg A.png foo.fimg
error in 'fimg_create_from_png' : read png -> -1 File error
png2fimg : err -1, abort.
```

Il faut peut-être envisager le passage à `libpng`.

6.3 fimgstats

Affichage de quelques valeurs calculées à partir du contenu d'un fichier `.fimg`.

```
usage : fimgstats [options] file.fimg
        -c      make a machinable csv
        -v      increase verbosity
```

À vrai dire, je ne sais pas encore quelles métriques seront utiles en première approche, alors commençont par le plus simple, les valeurs moyennes de chaque composante.

Puis nous rajouterons [17] le calcul de la variance. Les compétences de `schmod777` sont attendues au dd2.

17. Les patchs sont les bienvenus

6.4 fimgfx

Ce programme, *en cours de création*, applique un effet spécial à une image. À l'heure actuelle [18], nous avons déja quelques ajustements basiques de contraste, qui ne tiennent pas vraiment compte du contenu de l'image.

```
tth@daubian:~/Devel/FloatImg/tools$ ./fimgfx -v -h
--- fimg special effects ---
        cos01 cos010 pow2 sqrt gray0 xper
```

Certaines de ces opérations ont besoin d'un paramètre flottant. Celui-ci peut être fixé avec l'option `-k`. Une liste détaillée des opérations possibles sera lisible avec le sélecteur `-L`.

Ajustements de contraste : cos01 cos010 pow2 sqrt

Distorsions chromatiques : gray0

Déformations géométriques : r90

6.5 fimgops

Quelques opérations diverses entre deux images, qui doivent être de la même taille, et uniquement du type *RGB*. Certaines de ces opérations peuvent avoir un effet étrange sur vos images, par exemple si un pixel se retrouve avec une valeur négative.

```
usage:
        fimgops [options] A.fimg B.fimg operator D.fimg
operators:
        add             1
        sub             2
        mix             3
        mul             4
        mini            5
        maxi            6
options:
        -g              convert output to gray
        -k N.N          set float value
        -v              increase verbosity
        -X              explosive action
```

Pour des operateurs paramétrable (comme `mix`), le paramêtre flottant doit être fourni en utilisant l'option `-k`. La véracité mathématique n'est pas garantie. Et n'oubliez pas que les valeurs négatives peuvent être la cause de *glitches* de qualitay.

6.6 fimg2png, fimg2pnm, fimg2tiff, fimg2fits

Quelques petits proggies pour exporter notre format secret vers des choses plus directement utilisables. À condition que le code soit écrit et documenté.

D'un autre coté, écrire un greffon d'import/export pour Gimp ou ImageMagick ou Krita ne devrait pas être trop difficile. Des volontaires ?

D'ailleurs, pourquoi N logiciels indépendants alors q'un seul devrait être nécessaire ?

18. janvier 2019, vers 13 :37

6.7 fimg2gray

Nous avons vu dans ce document que chaque image flottante pouvait avoir plusieurs plans de réalité. Il ne faut en négliger aucun.

Il faut quand même deviner que pour passer de l'espace RGB à une abstraction linéaire monodimensionnelle, il existe une foultitude de méthodes, toutes plus légitimes que les autres.

7 TODO

Il reste plein de choses à faire pour que ce soit vraiment utilisable, surtout dans un contexte artistique à grande porosité. C'est par ces frottements de techniques ayant du sens que les choses seront acquises.

- ✷ Import/export au format TIFF.
- ✷ Remplacer le « fait-maison » par LIBNETPNM. *[en cours]*.
- ✷ Compléter les traitements mathémathiques (eg le gamma).
- ✷ Formaliser les codes d'erreur. **Urgent**.
- ✷ Faire une passe complète de Valgrind.
- ✷ Integrer la fonderie et l'interpolator.

8 Exemples pour yusers

Nous allons *essayer d'improviser* un exemple presque réel, avec un peu de rache dedans, et beaucoup de simplification. Ce qui est autorisé dans les exemples, mais dans la vrai vie, il ne faut jamais négliger le traitement des éventuelles erreurs.

Nous savons générer une image contenant des pixels aux valeurs probablement aléatoires, avec la commande `mkfimg`, qui utilise le `drand48` de POSIX. Maintenant, posons-nous une question de statisticien : ue se passe-t-il si nous faisons la somme de plusieurs centaines de ces images ?

```
#!/bin/bash
ACCU="quux.fimg"
TMPF="tmp.fimg"
DIMS="320 240"
mkfimg $ACCU $DIMS
for i in {0..1000}
do
        mkfimg -t drand48 ${TMPF} ${DIMS}
        fname=$( printf "xx%04d.pnm" $i )
        fimgops $ACCU $TMPF add $ACCU
        fimg2pnm -v -g $ACCU $fname
done
convert -delay 10 xx*.pnm foo.gif
```

Voilà, si les choses se passent mal, vous allez découvrir que votre `drand48` n'est pas si "drand" que ça. Et ce n'est pas à moi d'en tirer les conclusions...

8.1 Scripts

Le script bash `scripts/shoot.sh` est un front-end encore un peu rudimentaire vers le programme de capture d'image décrit page 18. Il utilise deux fichiers dans le répertoire de travail :

reglages et *compteur*. Le premier est, en fait, un bout de shell affectant quelques variables, ou plutôt, les surchargeant.

Voici un exemple de réglage :

```
OPTIONS="${OPTIONS} -v -c pow2 "
SHOW="yes"
NBRE=1000
PERIOD=0
OFORMAT="p_%04d.png"
```

La première ligne demande, en plus des options par défaut, plus de bavardage, et un changement de contraste. La seconde demande l'affichage de la photo. Les deux suivantes demandent la capture de 1000 images à la cadence méga-blast. La dernière est moins simple : `man sprintf` pour comprendre.

Quand au second fichier, il contient un compteur (stocké en ascii) qui est incrémenté après chaque capture réussie. Et ce compteur est utilisable par la variable `OFORMAT` que nous avons vue quelques lignes plus haut.

```
#!/bin/bash


# change this to point to your installed binary
#
GVS=${HOME}/Devel/FloatImg/v4l2/grabvidseq

#       ------------------------------------
#       set some default values
DEV=/dev/video2
SZ=640x480
NBRE=320
PERIOD=0.0
COUNT=compteur
OPTIONS="␣-v␣"
SHOW="no"
# output format can be of those types:
#       .pnm .fimg or .png
OFORMAT="P_%04d.pnm"

#               ces parametres peuvent etre surcharges avec
#               un fichier nomme "reglages" dans le repertoire
#               de travail.

#       ------------------------------------
#       overide parameters from $PWD
if [ -r ./reglages ]
then
        source ./reglages
fi

#       ------------------------------------
#       get the current picture number
if [ -r $COUNT ]
then
        numero=$( head -1 $COUNT )
else
        numero=1
fi
```

```
#       ------------------------------------
#       make the output filename
if [ 1 -eq $# ]
then
        outfile="$1"
else
        outfile=$( printf ${OFORMAT} $numero )
fi

#       ------------------------------------
#       grab and display the fancy picture
$GVS -d $DEV -n $NBRE -p $PERIOD $OPTIONS -s $SZ -o $outfile

if [ ${SHOW} == "yes" ]
then
        display $outfile &
fi

#       ------------------------------------
#       increment and save the picture number
numero=$(( numero + 1 ))
echo $numero > $COUNT
```

8.2 Fonderie

Ce projet externe [19] est destiné à la confection de films flous à partir de photos floues. Le script `scripts/echomix.sh` est une première expérimentation en bash, utilisant deux outils en CLI, le premier pouvant salement brotcher une image, et le second capable de mélanger harmonieusement deux images, la balance est équilibrée.

Il s'agit donc d'un petit programme écrit en Bash, un langage dont la connaissance est, pour moi, indispendable à qui veut faire des images kitchies. Mais ne vous inquiétez pas, c'est en fait assez simple à comprendre. Et comprendre, c'est apprendre.

Voici donc le script, décomposé et expliqué :

```
#!/bin/bash

SRCDIR="Fist"
DSTDIR="Pong"
FTMP="/dev/shm/tmp.fimg"
FDST="/dev/shm/foo.fimg"

# count the nomber of picz in the source directory
NBRE=$(ls -1 ${SRCDIR}/*.fimg | wc -l)
# compute the echo picz offset
OFFS=$(( NBRE / 4 ))
```

Dans ce préliminaire logiciel, nous avons nommés le répertoire SRCDIR contenant les captures d'image au format fimg, le répertoire DSTDIR dans lequel seront rangées les images calculées, et l'emplacement de deux fichiers de travail.

Les quelques lignes suivantes, qui semble bien magiques, ne sont en fait que de la magie Unix. Elles nous permettent d'avoir NBRE, le nombre d'images à traiter, et OFFS, un décalage dépendant du nombre d'image. Muni de toutes ces informations, nous pouvons rentrer dans le lard du sujet, la boucle qui travaille.

19. ... pour le moment, j'ai des soucis sur l'architecture du **pipdeprod** à adopter...

```
#                                                MAIN LOOP
for idx in $(seq 0 $NBRE)
do
        # build the two input filenames ...
        #
        imgA=$(printf "$SRCDIR/%04d.fimg" $idx)
        vb=$(( $(( idx + OFFS )) % NBRE))
        imgB=$(printf "$SRCDIR/%04d.fimg" $vb)

        #        ... and the output filename
        #
        dst=$(printf "%s/%05d.png" ${DSTDIR} $idx)
```

Dans cette première partie de la boucle nous avons construit plusieurs noms de fichier à partir du rang de la boucle en cours d'exécution, des deux valeurs NBRE et OFFS calculées en préambule.

```
        # trying to autocompute the mixing coefficient
        #
        compute=" s(${idx} / 16) "
        K=$(echo $compute | bc -l)
        printf " %25s  =>  %8.3f\n" "$compute" $K
```

Cette seconde partie sert à calculer avec la commande `bc` un coefficient variable en fonction du temps : $sin(idx/16)$ afin d'avoir une oscillation du coefficient entre -1.0 et 1.0, deux valeurs probablement glitchantes.

```
        #        do the hard floating computation
        #
        fimgfx  -v cos010 ${imgB} ${FTMP}
        fimgops    -k ${K} ${FTMP} ${imgA} mix ${FDST}
```

Étape suivante, étape cruciale : le brassage d'une multitude de pixels flottants.

Tout d'abord, nous faisons subir à l'image-echo (`imgB`, définie au début du script) un distorsion chromatique de type *cos010*, le résultat étant écrit dans un fichier temporaire. Ensuite, nous mixons l'image primaire et son echo en utilisant le rapport de mixage calculé quelques lignes plus haut.

```
        #        write the output as PNG for video encoding
        #
        fimg2png ${FDST} ${dst}
done
```

Et en fin de boucle, nous convertissons le résultat de nos savants calculs au format PNG, et écrivons le fichier dans le répertoire de destination fixé au début. C'est le moment de passer la main à ffmpeg.

C'est juste une POC, et une implémentation bien plus complète écrite en **C** est déja en chantier, avec une complexité prévue à un niveau assez réjouissant.

9 Video for Linux

Donc, maintenant, nous savons un peu tripoter ces images flottantes. Et nous devons nous poser une question fondamentale [20] sur la provenance de ces données prétendant être des images.

20. primitive ?

En fait, notre désir secret (enfin, surtout le mien) est la découverte des choses cachées du monde qui nous entoure. Nous voulons des images du **réel** et pour cela, l'outil le plus commun, le plus répandu, est la webcam. L'universelle webcam. Et l'incontournable v4l2.

9.1 grabvidseq

Un logiciel en évolution (trop ?) lente, qui permet déja la capture d'images en *longue pose* selon la méthode du cumul, et devrait bientôt retrouver sa capacité à enregistrer des séquences d'images.

```
tth@debian:~/Devel/FloatImg/v4l2$ ./grabvidseq -h
options :
        -c quux         contrast adjustement
        -d /dev/?       select video device
        -g              convert to gray
        -n NNN          how many frames ?
        -O ./           set Output dir
        -o bla          set output filename
        -p NN.N         delay between frames
        -r 90           mode portrait
        -s WxH          size of capture
        -u              try upscaling...
        -v              increase verbosity
        -X arg          Xperiment option
```

La plupart de ces options ont un usage quasi-évident. L'option `-s` doit correspondre à une des résolutions possibles de votre capteur. Le type du fichier en sortie (option `-o`) est déterminé par l'extension du nom. Actuellement seulement `.fimg`, `.pnm`, `.fits`, `.tiff` et `.png` sont reconnus.

La conversion en gris (option `-g`) mérite un peu plus de travail, et une paramétrisation plus facile. L'ajustement de contraste (option`-c`) est vaguement expliqué page 7.

L'option `-X` me permet d'intégrer des *fritures* expérimentales dans le binaire, et ne doit donc pas être utilisée dans des scripts si on a des visions à long terme.

9.1.1 Upscaling

La fonction que j'ai appelée *upscaling* est un petit hack qui permet de doubler artificiellement la résolution de l'image, en profitant du fait que l'on est capable de prendre N images en rafale.

Pour être rigoureux dans la prise de vue, ce N doit être un multiple de 4, surtout si le nombre de capture est faible. N'hésitez pas à faire des essais, le résultat est parfois aléatoire, surtout avec une caméra qui bouge.

Là, il manque un schéma...

9.2 video-infos

Que contient, que peut faire mon périphérique *àlc* ? Quelles sont ses possibilités de réglage ?

```
tth@debian:~/Devel/FloatImg$ v4l2/video-infos -h
Options :
        -d              select the video device
        -K nnn          set the K parameter
        -l              list video devices
        -T bla          add a title
        -v              increase verbosity
```

Je me sens obligé d'avouer qu'il reste quelques points mystérieux dans l'API de V4L2, et donc, que ce que raconte ce logiciel doit être pris avec des pincettes. En particulier la liste des résolutions disponibles.

9.3 nc-camcontrol

Ajustement *Brightness Contrast Saturation Hue...*

10 À l'extérieur

10.1 ImageMagick

Pour afficher notre format .fimg exotique avec `display`, vous devez mettre ce bout de XML dans le fichier `$HOME/.magick/delegates.xml` :

```
<?xml version="1.0" encoding="UTF-8"?>
<delegatemap>
    <delegate decode="fimg" command="fimg2png '%i' '%o'"/>
</delegatemap>
```

C'est juste un hack rapide, qui ne fonctionne pas très bien avec d'autres commande de IM, comme identify, qui a tendance à raconter un peu n'importe quoi... Je compte donc sur le bouquin de *Brunus* pour avancer...

10.2 Gimp

Mmmmm... Ça semble un peu plus compliqué. La documentation à ce sujet me semble ésotérique. D'un autre coté, il faut faire ça en **C**, ce qui ne peut être négatif.

10.3 ffmpeg

Un petit aide-mémoire pour encoder vos superbes timelapses :

```
ffmpeg  -nostdin                                \
        -y -r 30 -f image2 -i stereo/S%04d.png  \
        -c:v libx264                            \
        -pix_fmt yuv420p                        \
        -tune film                              \
        stereo.mp4
```

10.4 Et encore ?

Il y a d'autres logiciels pour lesquels écrire une fonction d'importation serait bien : *Geeqie*, un visualiseur d'image fort pratique, ou *Krita* qui semble avoir les faveurs de dessinateurs de talent.

11 Et pour la suite ?

En fait, je fait de la photo par la méthode du « cumul » depuis plusieurs années. Une webcam, un Linux, et ça *juste marche*. Sauf que c'est quand même un peu galère à déplacer, il faut avoir un shell pour déclencher, c'est pas facile à utiliser en mode portnawak...

L'idée est donc de construire un appareil autonome, basé sur un Raspi et une webcam USB, pilotable par LIRC, alimenté par une (grosse) batterie et permettant d'aller faire des images au bord d'un lac ou dans la campagne de l'Ariège.

Index